DANIEL LOPES RODRIGUES

IMPLEMENTAÇÃO DE ALGORITMOS PARA O CONTROLE SUPERVISÓRIO MODULAR DE SISTEMAS CONDIÇÃO/EVENTO

FLORIANÓPOLIS
2004

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

IMPLEMENTAÇÃO DE ALGORITMOS PARA O CONTROLE SUPERVISÓRIO MODULAR DE SISTEMAS CONDIÇÃO/EVENTO

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

DANIEL LOPES RODRIGUES

Florianópolis, maio de 2004.

IMPLEMENTAÇÃO DE ALGORITMOS PARA O CONTROLE SUPERVISÓRIO MODULAR DE SISTEMAS CONDIÇÃO/EVENTO

Daniel Lopes Rodrigues

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica, Área de Concentração em *Controle, Automação e Informática Industrial*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

José Eduardo Ribeiro Cury, Dr. D' Etat.
Orientador

Jefferson Luiz Brum Marques, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

José Eduardo Ribeiro Cury, Dr. D' Etat.
Presidente

Eduardo Camponogara, Ph.D

Romulo da Silva Oliveira, Dr.

Victor Juliano De Negri, Dr. Eng.

Dedico este trabalho a todas as pessoas que sempre acreditaram em mim.

AGRADECIMENTOS

Agradeço a Deus pela minha existência, por estar sempre presente em todos os momentos, iluminando meu caminho, e por ter me presenteado com pessoas maravilhosas os quais, contribuíram muito para minhas conquista, de forma direta ou indireta.

Agradeço a minha família, Pai, Mãe, Avó e Irmão que me deram muito apoio, estando presente sempre, nas horas boas e ruins. Um agradecimento especial a minha amada Juliana Hartmann e família.

Agradeço ao meus orientadores José Cury, André Leal e colegas de pesquisa pelo carinho e paciência, os quais foram pessoas de grande importância para meu sucesso, me ajudando crescer muito como pessoa e como estudante. E nunca vou me esquecer deles.

Agradeço muito a todos meus amigos pela amizade, carinho, e por sempre acreditarem que tudo daria certo, sendo assim pessoas de grande importância para o meu sucesso. Pois de forma direta ou indireta sempre me ajudaram, e nunca vou me esquecer dessas pessoas maravilhosas. Em especial ao meu amigo Allan, que infelizmente partiu antes da realização deste trabalho.

Agradeço a CAPES pelo apoio financeiro.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

IMPLEMENTAÇÃO DE ALGORITMOS PARA O CONTROLE SUPERVISÓRIO MODULAR DE SISTEMAS CONDIÇÃO/EVENTO

DANIEL LOPES RODRIGUES

Maio/2004

Orientador: José Eduardo Ribeiro Cury
Área de Concentração: Controle, Automação e Informática Industrial
Palavras-chave: Controle supervisório; sistemas condição/evento; síntese de supervisores; sistemas a eventos discretos; controle modular.
Número de Páginas: xiv + 70

O presente trabalho visa o desenvolvimento e implementação de algoritmos envolvidos na síntese de supervisores modulares para sistemas modelados através do paradigma de sistemas condição/evento (C/E). Os sistemas condição/evento consistem em uma classe de sistemas a eventos discretos em tempo contínuo, que permitem a modelagem e a análise de sistemas de tempo contínuo como a interconexão de subsistemas com sinais de entrada e saída discretos. Nesta abordagem, pode-se modelar o sistema de modo que os eventos sejam associados a saídas da planta(por exemplo, sinais de sensores) e os comandos gerados pelo agente de controle sejam associados a entradas da planta(condições), o que permite obter modelos cujos sinais de entrada e saída sejam mais próximos aos reais. Além disto, ela se adapta perfeitamente tanto à modelagem de sistemas a eventos discretos (SEDs) quanto à modelagem de sistemas híbridos (SHs) e possibilita que a modelagem destes sistemas seja feita através de diagramas de blocos e de fluxos de sinais, tal como ocorre normalmente na teoria de sistemas. O problema de controle supervisório modular é formulado através do paradigma de modelagem de sistemas C/E, e é resolvido através da teoria de controle supervisório de SEDs. Baseado no contexto da metodologia proposta, algoritmos foram desenvolvidos e implementados, formando uma ferramenta computacional que reúne as funções necessárias para a síntese de supervisores para sistemas modelados através deste paradigma. Um exemplo prático é utilizado para ilustrar a utilização da ferramenta para a resolução de um problema de controle supervisório modular.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

IMPLEMENTATION OF ALGORITHMS FOR THE MODULAR SUPERVISORY CONTROL OF CONDITION/EVENT SYSTEMS

DANIEL LOPES RODRIGUES

May/2004

Advisor: José Eduardo Ribeiro Cury
Area of Concentration: Control, Automation and Industrial Computing
Key words: supervisory control; condition/event systems; synthesis of supervisors; discrete-event systems; modular control.
Number of Pages: xiv + 70

The present work concerns the development and implementation of algorithms involved in the modular synthesis of supervisors for systems modeled through the paradigm of condition/event (C/E) systems. The condition/event systems consist of a class of continuous-time discrete event systems that provide a framework for modeling and analyzing continuous-time systems as the interconnection of subsystems with discrete input and output signals. In this approach, we can model the system in such a way that the events are associated to outputs of the plant(sensors signals, for instance) and the commands generated by the control agent are associated to inputs of the plant (conditions), what provides an intuitive modeling framework amenable to block diagram representation with input and output signals that are close to the real ones. Furthermore, this formalism is suitable for modeling both discrete event systems and also hybrid systems. The modular supervisory control problem is formulated within the C/E formalism and is solved through the theory of supervisory control for discrete event systems. Based on the context of the proposed methodology, algorithms were developed and implemented in order to build a computational toolbox for the modular synthesis of supervisory control for condition/event systems. A practical modular supervisory control problem is solved in order to illustrate the use of this toolbox.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

A tecnologia moderna tem gerado sistemas cada vez mais complexos que, seja pela importância que adquirem em seu contexto, seja por sua complexidade e seu custo, ratificam o enorme esforço em sua modelagem, análise, otimização e automação. Tais sistemas estão evidentes em uma série de aplicações, incluindo por exemplo a automação da manufatura, tráfego, comunicação, robótica, logística, sistemas operacionais, redes de computadores, gerenciamento de bases de dados, dentre outros [1, 19, 21].

A teoria de Sistemas a Eventos Discretos (SEDs) constitui uma abordagem formal à problemática deste tipo de sistema, caracterizando-os em modelos cuja dinâmica é conduzida por eventos instantâneos, que acontecem assincronamente. Por exemplo, um evento pode ser o início ou o término de uma tarefa, a percepção de uma mudança de estado de um sensor ou a pane de uma máquina em um sistema de manufatura. Um SED possui espaço de estados discretos onde cada estado é associado a valores simbólicos ou lógicos, dependendo da aplicação; assim as condições tais como "máquina ociosa, trabalhando ou quebrada" podem definir um estado. Os estados de um SED mudam conforme a ocorrência assíncrona e instantânea de eventos. A natureza discreta dos SEDs faz com que os modelos matemáticos convencionais, baseados em equações diferenciais, não sejam adequados para tratá-los. Por outro lado, a sua importância faz com que seja altamente desejável encontrar soluções para problemas relacionados com seu controle. Em razão disso, existe uma intensa atividade de pesquisa voltada à busca de modelos matemáticos apropriados à sua representação, sem que se tenha conseguido até agora achar um modelo que seja matematicamente tão conciso e computacionalmente tão adequado como são as equações diferenciais para sistemas dinâmicos de variáveis contínuas [21]. Dentre os modelos existentes, destaca-se o proposto por Ramadge e Wonham, o qual é baseado na Teoria de Linguagens e Autômatos e denominado "modelo RW".

Uma outra abordagem à problemática em questão constitui o estudo dos chamados Sistemas Híbridos (SHs). Em geral, os sistemas híbridos são sistemas que possuem tanto dinâmicas contínuas quanto dinâmicas discretas. Tais sistemas têm sido utilizados nas mais diversas aplicações: na avia-

ção, através do controle de tráfego aéreo e dos controles de sistemas de aeronaves, tais como sistemas de prevenção de colisões de aeronaves; em aplicações automotivas como, por exemplo, os sistemas de controle de tráfego em rodovias e os sistemas de controle de veículos inteligentes, entre muitas outras.

1.1 Motivação e Objetivos

Com efeito, no contexto atual, apresenta-se de fundamental relevância o problema de desenvolvimento de sistemas computacionais que interajam com sistemas modelados como SEDs, de forma a controlá-los.

Na teoria clássica de controle supervisório [19, 20], o sistema a ser controlado é concebido como um gerador espontâneo de eventos, denominado planta. Por outro lado, o agente de controle, chamado de supervisor, direciona a atuação da planta conforme um dado comportamento desejado. Seguindo este raciocínio, denota-se relevante o fato de que o conjunto de eventos é considerado como gerado apenas pela planta controlada, enquanto o controlador age de forma permissiva, e não como uma unidade que gera eventos. Reagindo à ocorrência de eventos gerados pela planta, o controlador deve habilitar ou desabilitar eventos seguintes da planta, assegurando uma trajetória do sistema nos moldes do comportamento desejado.

Na literatura, várias extensões vem sendo apresentadas para o "modelo RW". Dentre estas, os sistemas condição/evento podem ser vistos como uma classe de sistemas a eventos discretos em tempo contínuo, que permitem a modelagem do sistema como a interconexão de subsistemas com sinais de entrada e saída discretos. Nesta abordagem, pode-se modelar o sistema de modo que os eventos sejam associados a saídas da planta (por exemplo, sinais de sensores) e os comandos gerados pelo agente de controle sejam associados a entradas da planta(condições). Assim, em muitos casos, esta metodologia de modelagem mostra-se mais intuitiva e natural que aquela baseada no modelo RW. Além disto, ela possibilita que a modelagem de SEDs baseie-se em diagramas de blocos e em fluxos de sinais, tal como ocorre normalmente na teoria de sistemas. É importante ressaltar ainda que este paradigma de modelagem adapta-se também à modelagem de sistemas híbridos, os quais são objeto de estudo do trabalho do doutorando André Bittencourt Leal, cujo tema é o "Controle Supervisório Modular de Sistemas Híbridos" [11].

Para a solução de um problema de controle supervisório, duas abordagens podem ser utilizadas: a abordagem monolítica e a abordagem modular. Na abordagem monolítica sintetiza-se um único supervisor para garantir o cumprimento das especificações. Na abordagem de controle modular [1], ao invés de se projetar um único controlador que satisfaça todas as especificações, procura-se construir um supervisor para cada especificação, de forma que, atuando em conjunto, satisfaçam a especificação global. Em geral, o emprego da abordagem de controle modular é motivada por dois fatores: maior

flexibilidade e menor complexidade computacional. Por exemplo, se uma subtarefa é alterada, só é preciso reprojetar o controlador correspondente ao invés de se refazer todo o projeto do supervisor.

Deve-se salientar ainda que, em geral, os modelos obtidos para representar o comportamento lógico de sistemas a eventos discretos de grande porte, bem como o de sistemas híbridos, são modelos complexos (com grande número de estados e transições). Entretanto, o aumento da complexidade do modelo leva a uma maior dificuldade na síntese de supervisores, tornando imprescindível a automatização do processo de obtenção destes supervisores. Este fato motivou o desenvolvimento do presente trabalho, que tem como objetivo o desenvolvimento de algoritmos envolvidos na síntese de supervisores modulares para sistemas modelados através do paradigma de sistemas condição/evento (C/E), bem como na implementação destes em uma ferramenta computacional denominada Grail[1][15]. Deseja-se, então, suprir o Grail com um pacote de funções para o controle supervisório modular de sistemas condição/evento. Deve-se ressaltar que estas funções são utilizadas no contexto do trabalho de doutorado do aluno André B. Leal, citado anteriormente.

1.2 Estrutura da dissertação

O restante deste trabalho está organizado como segue:

O Capítulo 2 apresenta os sistemas C/E [10, 17]. Nele é feita a formulação de um problema de controle supervisório para sistemas a eventos discretos, bem como a formulação de um problema de controle supervisório para sistemas híbridos. A formulação destes problemas é feita através do formalismo de sistemas condição/evento, mas a resolução de cada um deles é obtida através da resolução de um problema equivalente formulado no domínio de SEDs. Um exemplo bastante simples é utilizado de forma a ilustrar a metodologia de resolução apresentada.

No Capítulo 3 trata-se da abordagem modular de síntese de supervisores para sistemas C/E. De forma análoga ao Capítulo 2, formula-se o problema de controle através do paradigma de modelagem de sistemas condição/evento, mas sua resolução é feita através de um problema equivalente no domínio de SEDs. O exemplo utilizado no Capítulo 2 é retomado para a resolução através da abordagem modular, permitindo então a comparação dos resultados obtidos.

No Capítulo 4 estão concentradas as maiores contribuições deste trabalho. Nele serão apresentados os algoritmos envolvidos na solução do problema de síntese de supervisores modulares para sistemas C/E sob o ponto de vista da teoria de controle supervisório de SEDs.

No Capítulo 5 apresenta-se um exemplo prático de controle supervisório e desenvolve-se toda a resolução do problema com auxílio da ferramenta computacional Grail e das funções que implementam os algoritmos desenvolvidos ao longo deste trabalho, ilustrando assim a utilização das mesmas.

[1]O grail é uma biblioteca de funções desenvolvida em C++, que permite a computação simbólica sobre máquinas de estado finitas, expressões regulares e linguagens

Finalmente no Capítulo 6, apresentam-se as conclusões da pesquisa, bem como sugestões para trabalhos futuros.

Capítulo 2

Sistemas Condição/Evento

Este capítulo tem como objetivo apresentar os sistemas Condição/Evento (C/E) como um paradigma de modelagem que servirá de base tanto para a modelagem de Sistemas a Eventos Discretos (SEDs), como para a modelagem de Sistemas Híbridos (SHs). Apresenta-se ainda a teoria de controle supervisório para sistemas C/E.

2.1 Introdução

Os Sistemas C/E [10, 17], são sistemas a eventos discretos em tempo contínuo com duas classes de sinais: os sinais condição, constantes por partes e que toma valores de um conjunto finito de condições; e os sinais eventos, não nulos apenas em pontos discretos do tempo, quando assume valores sobre um conjunto finito de eventos. Em geral, ambos os tipos de sinais podem ser sinais de entrada e saída para sistemas C/E, como ilustrado na Figura 2.1 [10]. No formalismo de sistemas C/E, transições de estados podem ser habilitadas ou desabilitadas pelo sinal de entrada condição e forçadas pelo sinal de entrada evento.

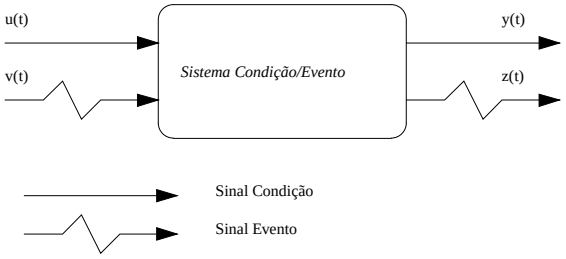


Figura 2.1: Sistema Condição/Evento

Uma das grandes motivações para o uso da abordagem de sistemas C/E, é que ela permite uma forma de se definir sistemas em tempo contínuo através da interconexão de subsistemas com sinais

de entrada e saída discretos. Além disto, ela se adapta perfeitamente tanto à modelagem de sistemas a eventos discretos, quanto à modelagem de sistemas híbridos e possibilita que a modelagem destas classes de sistemas seja baseada em diagramas de blocos e fluxos de sinais, como é feito normalmente na teoria de sistemas. Para esta primeira classe de sistemas, acredita-se que este tipo de paradigma de modelagem torna-se mais natural e intuitivo que aqueles oferecidos por formalismos estritamente discretos, tais como autômatos e linguagens formais.

2.2 Definições

Esta seção tem o intuito de apresentar os sistemas C/E. Todas as definições e teoremas foram baseados em [17].

2.2.1 Sistemas e Sinais Condição/Evento

A descrição de entrada e saída de um Sistema C/E é baseada nas duas classes de sinais apresentadas a seguir.

Definição 2.1 (Sinal Condição) *A função $x : [0,\infty) \to X$, é um sinal condição sobre X em [0, ∞) se $x(\cdot)$ é um sinal contínuo à direita e com limite à esquerda.*

Definição 2.2 (Sinal Evento) *A função $x : [0,\infty) \to X$, é um sinal evento sobre X em [0, ∞) se para qualquer intervalo de tempo $[t_1,t_2]$ ∈ [0, ∞) o conjunto $\{t \in [t_1,t_2] : x(t) \neq 0\}$ é finito.*

A Figura 2.2 ilustra os sinais evento e condição.

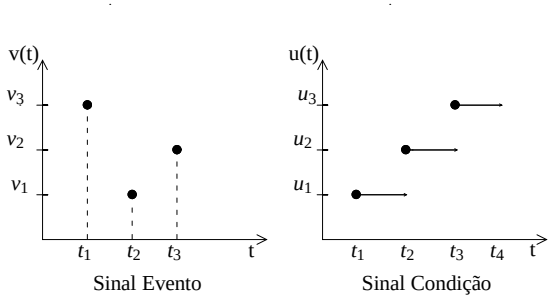


Figura 2.2: Sinais evento e condição

O conjunto de todos os sinais condição ou evento sobre X em [0, ∞) é referenciado por $\mathcal{X}$ (X caligráfico).

Definição 2.3 (Sistemas C/E) *Dados os conjuntos finitos e disjuntos U,V,Y e Z. Um sistema condição/evento em [0,∞) com entrada condição $\mathcal{U}$, entrada evento $\mathcal{V}$, saída condição $\mathcal{Y}$ e saída evento $\mathcal{Z}$, é um mapa $\mathcal{S} : \mathcal{U} \times \mathcal{V} \to 2^{\mathcal{Y} \times \mathcal{Z}}$ tal que, para cada entrada $(u(\cdot), v(\cdot)) \in \mathcal{U} \times \mathcal{V}$ existe pelo menos uma saída $(y(\cdot), z(\cdot)) \in \mathcal{Y} \times \mathcal{Z}$ tal que, $(y(\cdot), z(\cdot)) \in \mathcal{S} : (u(\cdot), v(\cdot))$.*

Em palavras, um sistema C/E identifica um conjunto de saídas $(y(\cdot), z(\cdot))$ com cada entrada $(u(\cdot), v(\cdot))$. Assim, em geral, sistemas C/E são não determinísticos.

A seguir, algumas propriedades são definidas para estabelecer classes úteis de sistemas C/E. As propriedades são: causalidade, invariância na escala do tempo e espontaneidade.

- Causalidade : Um sistema C/E é dito causal quando dadas duas entradas iguais até um certo instante de tempo, assegura-se que há saídas (para as respectivas entradas) que também são iguais até o mesmo instante de tempo;
- Invariância na escala de tempo : Um sistema C/E possui invariância na escala do tempo quando classes de equivalência de pares admissíveis de sinais de entrada e saída do sistema podem ser identificadas, independentemente da escala de tempo;
- Espontaneidade : Um sistema C/E é dito espontâneo quando os sinais de entrada condição podem apenas inibir ou habilitar os possíveis sinais de saída evento, mas a mudança no sinal de entrada condição não força a ocorrência de um sinal evento de saída.

A seguir será apresentado o modelo de estado discreto para o sistema C/E, mas antes algumas definições preliminares são necessárias.

Definição 2.4 (Sinal de entrada síncrono) *Uma entrada $(u(\cdot), v(\cdot)) \in \mathcal{U} \times \mathcal{V}$ é dita ser uma entrada síncrona se $\forall t \in [0,\infty), u(t^-) \neq u(t) \Rightarrow v(t) \neq 0$.*

Assim, uma entrada $(u(\cdot), v(\cdot)) \in \mathcal{U} \times \mathcal{V}$ é sincrona se mudanças no sinal condição $u(\cdot)$ ocorrem somente nos instantes de descontinuidade do sinal evento $v(\cdot)$.

Definição 2.5 (Sistema C/E em tempo discreto) *Um sistema C/E $\mathcal{S}$ é um sistema C/E em tempo discreto se o conjunto de entrada evento é não vazio e para qualquer entrada síncrona $(u(\cdot), v(\cdot)) \in \mathcal{U} \times \mathcal{V}$ e qualquer saída $(y(\cdot), z(\cdot)) \in \mathcal{S}(u(.), v(.))$ a seguinte condição é satisfeita: $\forall t \in [0,\infty), y(t^-) \neq y(t)$ ou $z(t) \neq 0 \Rightarrow v(t) \neq 0$.*

Assim, em um sistema C/E em tempo discreto, mudanças no valor do sinal de saída condição e descontinuidades no sinal de saída evento só ocorrem quando existirem descontinuidades no sinal de entrada evento.

Definição 2.6 (Sistema C/E determinístico) *Um sistema C/E $\mathcal{S}$ é determinístico se para cada entrada $(u(\cdot), v(\cdot))$ existe apenas uma saída $(y(\cdot), z(\cdot)) = \mathcal{S}(u(\cdot), v(\cdot))$.*

2.2.2 Modelo C/E e Realização de Estado Discreto

Segundo [17], raramente define-se um sistema C/E explicitamente em termos do conjunto de sinais de saída admissíveis para cada sinal de entrada. Normalmente, o mapeamento de entrada-saída é definido pelo Modelo C/E apresentado a seguir. Além disto, para se estudar propriedades de sistemas C/E que sejam úteis para aplicações reais é necessário restringir a atenção para classes de sistemas com uma estrutura ou modelo particular [17].

Definição 2.7 (Modelo C/E) *Dado um conjunto U,V,Y,Z um modelo C/E $\mathcal{M}$ é um conjunto de equações tal que para cada entrada $(u(\cdot),v(\cdot)) \in \mathcal{U} \times \mathcal{V}$ existe pelo menos uma saída $(y(\cdot),z(\cdot)) \in \mathcal{Y} \times \mathcal{Z}$ que satisfaça as equações para $\mathcal{M}$.*

Um modelo C/E $\mathcal{M}$ implicitamente define um sistema C/E, denotado por $\mathcal{S}_{\mathcal{M}}$. Dado um sistema C/E $\mathcal{S}$, se $\mathcal{S} = \mathcal{S}_{\mathcal{M}}$, então diz-se que $\mathcal{M}$ é uma realização de $\mathcal{S}$.

Definição 2.8 (Modelo de Estado Discreto ou Realização de Estado Discreto) *Uma realização de estado discreto para o sistema C/E $\mathcal{S} : \mathcal{U} \times \mathcal{V} \to 2^{\mathcal{Y} \times \mathcal{Z}}$ é uma quíntupla (X,f,g,h,x_0) onde:*

- *X é um conjunto enumerável de estados;*
- *$f : X \times U \times V \to 2^X - \emptyset$ é a função de transição de estado, satisfazendo $x \in f(x,u,0), \forall x \in X$ e $\forall u \in U$. Ou seja, o sistema deve estar apto a permanecer em qualquer estado se não houverem sinais de entrada evento;*
- *$g : X \times U \to Y$ é a função de saída condição;*
- *$h : X \times X \times V \to Z$ é a função de saída evento, satisfazendo $h(x,x,0) = 0$ para todo $x \in X$, ou seja, a saída evento só será diferente de 0 se houver uma transição de estado;*
- *x_0 é o estado inicial.*

Um sinal de entrada $(u(\cdot),v(\cdot)) \in \mathcal{U} \times \mathcal{V}$, um sinal de saída $(y(\cdot),z(\cdot)) \in \mathcal{Y} \times \mathcal{Z}$ e uma trajetória de estados $x(\cdot) \in \mathcal{X}$ satisfazem o modelo de estado discreto quando $x(t_0) = x_0$ e as seguintes equações são satisfeitas para todo $t > t_0$:

$$\begin{aligned} x(t) &\in f(x(t^-),u(t^-),v(t)) \\ y(t) &= g(x(t),u(t)) \\ z(t) &= h(x(t^-),x(t),v(t)) \end{aligned}$$

Teorema 2.1 *Se o sistema C/E $\mathcal{S}$ tem uma realização de estado discreto, então $\mathcal{S}$ é causal, invariante em escala de tempo e espontâneo.*

2.2.3 Linguagem

Uma vez que um sistema C/E possui sinais de entrada e saída que assumem valores discretos no tempo, é possível associar a este um comportamento lógico, o qual será representado por linguagens de palavras de comprimento finito [17].

Deve-se ressaltar que no decorrer deste trabalho será utilizado um modelo simplificado, o qual possui como entrada apenas sinais condição $u(\cdot)$ e como saída apenas sinais evento $v(\cdot)$, conforme mostrado na Figura 2.3. Sendo assim, a definição de linguagem de sistemas C/E será apresentada com base neste modelo.

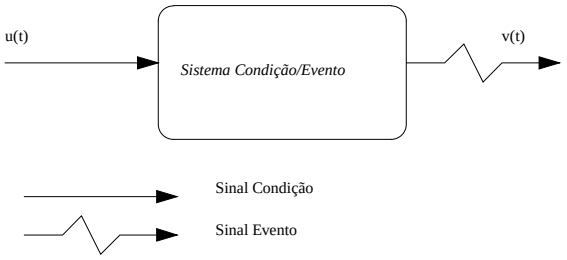


Figura 2.3: Modelo Simplificado

A linguagem de um sistema C/E é definida pelas seqüencias dos valores que os sinais eventos e condições assumem em seus pontos de descontinuidade, sem registrar no entanto os instantes de ocorrência de tais descontinuidades. Vale lembrar que os sinais $v(\cdot)$ e $u(\cdot)$ assumem valores sobre os conjuntos discretos V e U, respectivamente.

Conforme visto anteriormente, sinais condição são atualizados somente nos instantes de ocorrência de eventos. Desta forma, para possibilitar a representação da aplicação de um sinal condição inicial (antes da ocorrência de qualquer sinal evento) associa-se a este um evento fictício, denominado evento de inicialização, denotado por η. Portanto, η não está associado a transições de estado na planta, logo $\eta \notin V$. O conjunto $V^+ = \{\eta\} \cup V$ denota a inclusão do evento η em V. Sendo assim, para descrever os comportamentos lógicos dos sistemas C/E são utilizadas linguagens em $(V^+ \times U)^*$, ou mais especificamente em $(\{\eta\} \times U)(V \times U)^*$, uma vez que eventos η, são considerados apenas na inicialização da planta.

Seja, por exemplo, um sistema $\mathcal{S} \subseteq (V^+ \times U)^*$ com uma entrada condição $u(t)$ assumindo valores sobre o conjunto discreto $U = \{u_1, u_2, u_3\}$ e uma saída evento $v(t)$ assumindo valores sobre o conjunto discreto $V = \{v_1, v_2, v_3\}$. Considere que a seqüencia de sinais $(v(t), u(t)) \in (V^+ \times U)^*$ ilustrada na Figura 2.4 representa um possível comportamento temporal para o sistema, ou seja, $(v(t), u(t)) \in \mathcal{S}$.

A semântica deste comportamento é a seguinte: no instante t_0, o sistema começa a ser operado (inicialização) e é aplicado um sinal condição de entrada u_1, o sistema evolui e no instante t_1 ocorre um evento v_2; neste instante é aplicado o sinal condição u_2 e o sistema continua a sua evolução até a

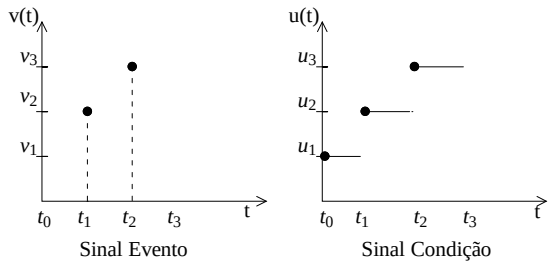


Figura 2.4: Comportamento possível para o Sistema $\mathcal{S}$

ocorrência do evento v_3 no instante t_2, quando é aplicado um sinal condição u_3, dando continuidade à operação do sistema. Podemos então ilustrar este comportamento lógico seqüencial pela cadeia $\eta u_1 \circ v_2 u_2 \circ v_3 u_3$. Na representação de cadeias o símbolo "$\circ$" denota a concatenação de pares $vu \in (V^+ \times U)$.

O comportamento lógico de um sistema C/E pode ser descrito por um par de linguagens: a linguagem gerada, que representa o comportamento fisicamente possível de ocorrer no sistema em termos de entrada e saída; e a linguagem marcada, que descreve o comportamento marcado do sistema, ou seja, corresponde às cadeias do sistema que possuem algum significado especial, como por exemplo, a finalização de tarefas. Observe que $\mathcal{L}_m(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{S}) \subseteq (\{\eta\} \cup U)(V \times U)^*$.

Para explorar o forte relacionamento entre sistemas C/E com realização de estado discreto e linguagens C/E, podemos utilizar modelos baseados em máquinas de estados finitas para representar o comportamento seqüencial lógico do sistema C/E.

A relação entre sistemas C/E com realização de estado discreto e linguagens representadas por máquinas de estados finitas é dado pelo Teorema 2.2.

Teorema 2.2 *Se um sistema C/E $\mathcal{S}$ tem realização de estado discreto, então existe uma máquina de estados finita para $\mathcal{L}(\mathcal{S})$, a linguagem de $\mathcal{S}$.*

A seguir são apresentados dois modelos de máquinas de estados finitas: autômatos de estados finitos e máquinas de Moore finitas. Mais informações sobre estes modelos, como linguagens representadas por estes, podem serem encontradas em [8], onde os modelos apresentados diferenciam dos apresentados aqui em termos dos alfabetos de entrada e saída.

Autômatos de Estados Finitos

Um autômato de estados finitos é uma quíntupla $G = (X, \Sigma, \delta, X_m, x_0)$, onde :

- X é o conjunto de estados finitos;

- Σ é o alfabeto ;
- $\delta : X \times \Sigma \to 2^X$ é a função de transição, possivelmente parcial e não determinística;
- X_m é o conjunto de estados marcados, $X_m \subseteq$ X;
- x_0 é o estado inicial.

Um autômato pode ser representado graficamente como um grafo orientado, onde os nós representam os estados e os arcos etiquetados representam as transições entre os estados. O estado inicial é identificado através de uma seta apontando para ele e os estados finais (marcados) são representados com círculos duplos.

A Figura 2.5 ilustra um exemplo simples de um autômato que modela o comportamento seqüencial lógico de um sistema C/E, cuja descrição formal é a seguinte:

- $X = \{x_0, x_1, x_2, x_3\}$ corresponde aos estados discretos do sistema;
- $\Sigma = \{\eta u_1, v_2 u_2, v_3 u_3\} \subseteq (V^+ \times U)$, onde $V^+ = \{\eta\} \cup \{v_2, v_3\}$ e $U = \{u_1, u_2, u_3\}$;
- A função de transição é dada por : $\delta(x_0, \eta u_1) = x_1$, $\delta(x_1, v_2 u_2) = x_2$, $\delta(x_2, v_3 u_3) = x_3$;
- $X_m = \{x_3\}$;
- $x_0 = \{x_0\}$.

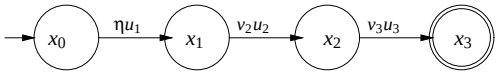


Figura 2.5: Autômato de estados finitos

O comportamento lógico de um sistema C/E pode ser modelado por um autômato G, onde $L(G)$ é o comportamento gerado pelo sistema e $L_m(G)$ é o comportamento marcado ou conjunto de tarefas completas do sistema.

Máquina de Moore Finita

Uma máquina de Moore (marcada) é uma sêxtupla $A = (X, I, O, \delta, X_m, x_0, \beta)$, onde:

- X é o conjunto de estados finitos;
- I é o alfabeto de entrada;
- O é um alfabeto de saída;

- $\delta : X \times I \rightarrow 2^X$ é a função de transição, possivelmente parcial e não determinística;
- X_m é o conjunto de estados marcados, $X_m \subseteq X$;
- $x_0 \subseteq X$ é o estado inicial ;
- $\beta : X \rightarrow O$ é a função de saída que associa a cada estado de x um símbolo de saída o .

2.3 Sistema Condição/Evento como Modelo para SEDs

Os sistemas C/E definem claramente sinais de entrada e saída, mostrando-se assim bastante interessantes para modelar sistemas a eventos discretos coordenados por CLPs (Controladores lógicos programáveis). Baseado nisto, apresenta-se uma metodologia de modelagem [6] para sistemas a eventos discretos cujos sinais podem ser classificados como sinais de entrada e saída, tendo como base o formalismo de sistemas C/E [17]. A partir disto, um pequeno exemplo é introduzido com o intuito de ilustrar a metodologia de modelagem.

2.3.1 Definição do Problema

Na linha de produção de uma indústria, existe uma mesa giratória na qual é feita a furação de peças. Esta mesa possui três posições, conforme mostrado na Figura 2.6, e seu funcionamento é comandado por um controlador lógico programável (CLP) conforme a seguinte seqüência de passos:

1. A esteira gira até que uma peça seja posicionada em P1;
2. A mesa gira 120^o;
3. A peça é furada;
4. A mesa gira 120^o;
5. O manipulador robótico retira a peça da mesa.

A mesa foi inicialmente projetada para operar em seqüência apenas uma peça por vez, ou seja, a esteira só pode ser acionada novamente depois que o manipulador retirar a peça da mesa. Esta restrição na lógica de controle evita os problemas que podem ocorrer na operação de múltiplas peças em paralelo, como por exemplo:

- operar a esteira, a furadeira, ou o manipulador enquanto a mesa estiver girando;
- sobrepor peças em P1;

- girar a mesa sem que as peças em P2 e P3 tenham sido furadas ou retiradas, respectivamente;
- furar ou acionar o manipulador sem peças nas posições P2 e P3, respectivamente;
- furar duas vezes a mesma peça;
- girar a mesa sem nenhuma peça.

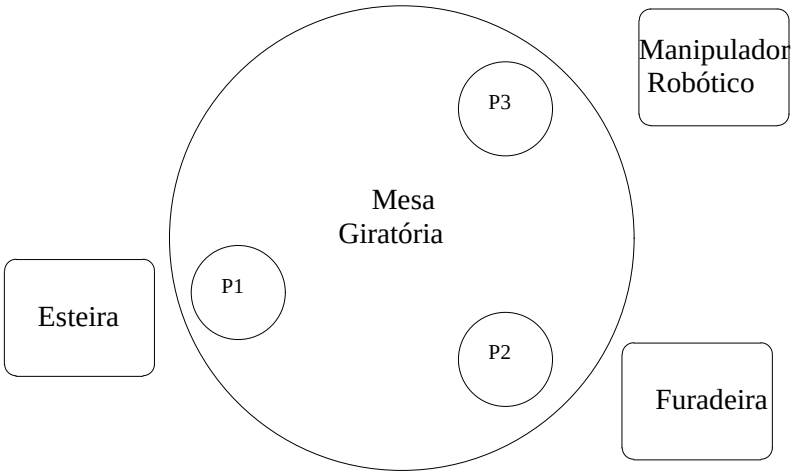


Figura 2.6: Sistema com mesa giratória

Entretanto, esse modo de funcionamento é pouco eficiente, visto que a esteira, a furadeira, e o manipulador passam a maior parte do tempo parados, enquanto poderiam estar operando em paralelo. O problema a ser resolvido consiste em desenvolver uma lógica de controle que garanta uma maior eficiência da mesa giratória e ao mesmo tempo evite os problemas citados anteriormente.

2.3.2 Modelagem do Sistema

O primeiro passo na modelagem do sistema consiste em encontrar os subsistemas envolvidos no sistema como um todo. Neste exemplo são quatro os subsistemas envolvidos: esteira, mesa giratória, furadeira e manipulador robótico.

O segundo passo consiste em verificar o relacionamento dos subsistemas e aplicar a operação de conexão sobre os subsistemas relacionados. Esta operação foi apresentada em [6, 7, 17]. Como neste exemplo os subsistemas não se relacionam, então as entradas condição e saídas evento são as entradas e saídas do sistema, respectivamente. Baseado nisto, o conjunto de saída condição e a função de saída condição não serão definidos.

O próximo passo é obter a realização de estado discreto para cada subsistema.

a) Esteira

O primeiro subsistema a ser analisado é a esteira (sistema C/E $\mathcal{S}_{est}$). A esteira possui uma realização de estado discreta $S_{est} = (X_{est},\ f_{est},\ h_{est},\ x_{est(0)})$ com os conjuntos V_{est}, U_{est}, onde :

- X_{est} é o conjunto de estados $X_{est} = \{e_m, e_p\}$;
- V_{est} é o conjunto de saída evento $V_{est} = \{v_{p_1}\}$;
- U_{est} é o conjunto de entradas condição $U_{est} = \{e^{on}, e^{off}\}$;
- f_{est} é a função de transição $f_{est} : X_{est} \times U_{est} \to X_{est}$ e está definido na Tabela 2.1 ;
- h_{est} é a função de saída evento $h_{est} : X_{est} \times X_{est} \to V_{est}^{+}$ e está definido na Tabela 2.2 ;
- $x_{est(0)}$ é o estado inicial da esteira, $x_{est(0)} = e_p$;

Tabela 2.1: Função de transição da esteira

x_{est}^{-}	u_{est}^{-}	x_{est}
e_m	e^{on}	e_m
e_m	e^{off}	e_p
e_p	e^{on}	e_m
e_p	e^{off}	e_p

Tabela 2.2: Função de saída evento da esteira

x_{est}^{-}	x_{est}	v_{est}
e_m	e_m	0
e_m	e_p	v_{p_1}
e_p	e_m	0
e_p	e_p	0

A esteira pode encontrar-se em dois estados distintos, em movimento (e_m) ou parada (e_p). A entrada condição e^{on} simboliza o comando de ligar a esteira e e^{off} o comando de desligar a esteira. O evento v_{p_1} sinaliza o final de operação da esteira. Utiliza-se a notação x_{est}^{-} e u_{est}^{-} para simbolizar o estado e a condição imediatamente anterior à transição.

Como apresentado em [6, 17] e citado anteriormente neste capítulo, a partir da realização de estado discreto de um sistema C/E, pode-se obter um autômato que representa o comportamento lógico seqüencial do mesmo. A Figura 2.7 ilustra o autômato para a esteira.

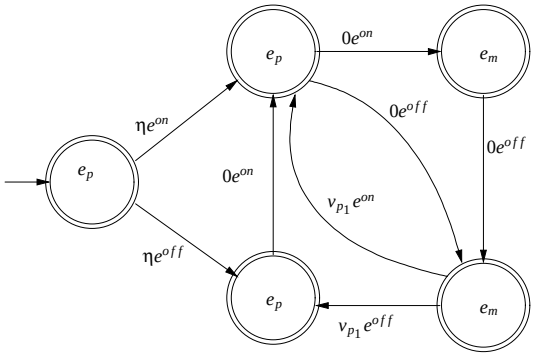


Figura 2.7: Autômato para a esteira S_{est}

b) Mesa

O subsistema mesa (sistema C/E $\mathcal{S}_{mesa}$) possui uma realização de estado discreta $S_{mesa} = (X_{mesa},\ f_{mesa},\ h_{mesa},\ x_{mesa(0)})$ com os conjuntos V_{mesa}, U_{mesa}, onde :

- X_{mesa} é o conjunto de estados $X_{mesa} = \{m_g, m_p\}$;
- V_{mesa} é o conjunto de saída evento $V_{mesa} = \{v_{120}\}$;
- U_{mesa} é o conjunto de entradas condição $U_{mesa} = \{m^{on}, m^{off}\}$;
- f_{mesa} é a função de transição $f_{mesa} : X_{mesa} \times U_{mesa} \rightarrow X_{mesa}$ e está definido na Tabela 2.3 ;
- h_{mesa} é a função de saída evento $h_{mesa} : X_{mesa} \times X_{mesa} \rightarrow V^{+}_{mesa}$ e está definido na Tabela 2.4 ;
- $x_{mesa(0)}$ é o estado inicial da mesa, $x_{mesa(0)} = m_p$;

Tabela 2.3: Função de transição da mesa

x^{-}_{mesa}	u^{-}_{mesa}	x_{mesa}
m_g	m^{on}	m_g
m_g	m^{off}	m_p
m_p	m^{on}	m_g
m_p	m^{off}	m_p

Tabela 2.4: Função de saída evento da mesa

x^{-}_{mesa}	x_{mesa}	v_{mesa}
m_g	m_g	0
m_g	m_p	v_{120}
m_p	m_g	0
m_p	m_p	0

A mesa pode encontrar-se em dois estados distintos, girando (m_g) ou parada (m_p). A entrada condição m^{on} simboliza o comando de ligar a mesa e m^{off} o comando de desligar a mesa. O evento v_{120} sinaliza o fim do giro de 120º da mesa. Na Figura 2.8 mostra-se o autômato que representa o comportamento lógico seqüencial da mesa.

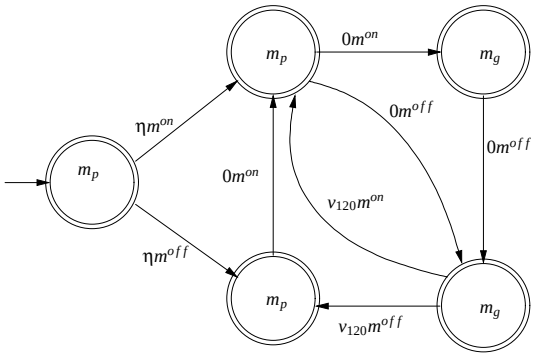


Figura 2.8: Autômato para a mesa S_{mesa}

c) Furadeira

O subsistema furadeira (sistema C/E $\mathcal{S}_{fur}$) possui uma realização de estado discreta $S_{fur} = (X_{fur},\ f_{fur},\ h_{fur},\ x_{fur(0)})$ com os conjuntos V_{fur}, U_{fur}, onde :

- X_{fur} é o conjunto de estados $X_{fur} = \{f_f, f_r\}$;
- V_{fur} é o conjunto de saída evento $V_{fur} = \{f_{fim}\}$;
- U_{fur} é o conjunto de entradas condição $U_{fur} = \{f^{on}, f^{off}\}$;
- f_{fur} é a função de transição $f_{fur} : X_{fur} \times U_{fur} \to X_{fur}$ e está definido na Tabela 2.5 ;
- h_{fur} é a função de saída evento $h_{fur} : X_{fur} \times X_{fur} \to V_{fur}^{+}$ e está definido na Tabela 2.6 ;
- $x_{fur(0)}$ é o estado inicial da furadeira, $x_{fur(0)} = f_r$;

Tabela 2.5: Função de transição da furadeira

x_{fur}^{-}	u_{fur}^{-}	x_{fur}
f_f	f^{on}	f_f
f_f	f^{off}	f_r
f_r	f^{on}	f_f
f_r	f^{off}	f_r

A furadeira pode encontrar-se em dois estados diferentes, furando (f_f) ou em repouso (f_r). A entrada condição f_{on} representa o comando de ligar a furadeira e f_{off} o comando de desligar a furadeira. O evento de resposta v_{fim} representa o término da operação de furação. A Figura 2.9, mostrada a seguir, ilustra o autômato para a furadeira.

Tabela 2.6: Função de saída evento da furadeira

x^{-}_{fur}	x_{fur}	v_{fur}
f_f	f_f	0
f_f	f_r	f_{fim}
f_r	f_f	0
f_r	f_r	0

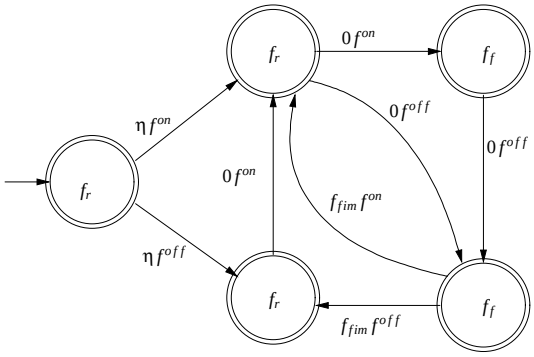


Figura 2.9: Autômato para a furadeira S_{fur}

d) Manipulador Robótico

O último subsistema a ser analisado é o manipulador robótico (sistema C/E $\mathcal{S}_{rob}$). O manipulador robótico possui uma realização de estado discreta $S_{rob} = (X_{rob},\ f_{rob},\ h_{rob},\ x_{rob(0)})$ com os conjuntos V_{rob}, U_{rob}, onde :

- X_{rob} é o conjunto de estados $X_{rob} = \{r_t, r_r\}$;
- V_{rob} é o conjunto de saída evento $V_{rob} = \{r_{fim}\}$;
- U_{rob} é o conjunto de entradas condição $U_{rob} = \{r^{on}, r^{off}\}$;
- f_{rob} é a função de transição $f_{rob} : X_{rob} \times U_{rob} \to X_{rob}$ e está definido na Tabela 2.7 ;
- h_{rob} é a função de saída evento $h_{rob} : X_{rob} \times X_{rob} \to V^{+}_{rob}$ e está definido na Tabela 2.8 ;
- $x_{rob(0)}$ é o estado inicial do manipulador, $x_{rob(0)} = r_r$;

O manipulador robótico possui dois estados discretos: transportando (r_t) e em repouso (r_r). As entradas condição r^{on} e r^{off} simbolizam os comandos de ligar e desligar o manipulador, respectivamente. O evento de saída r_{fim} representa o término da operação de transporte de peça pelo manipulador. O autômato que modela o comportamento do manipulador robótico é mostrado na Figura 2.10.

Tabela 2.7: Função de transição do manipulaor

x^{-}_{rob}	u^{-}_{rob}	x_{rob}
r_t	r^{on}	r_t
r_t	r^{off}	r_r
r_r	r^{on}	r_t
r_r	r^{off}	r_r

Tabela 2.8: Função de saída evento do manipulador

x^{-}_{rob}	x_{rob}	v_{rob}
r_t	r_t	0
r_t	r_r	r_{fim}
r_r	r_t	0
r_r	r_r	0

2.3.3 Modelo da Planta Livre

Após obter o modelo C/E para cada subsistema, pode-se obter o modelo para a planta livre $\mathcal{P}$. Este modelo é obtido através do algoritmo da operação empilhar, o qual foi apresentado em [6, 17] e implementado em [6] na linguagem C . Assim obtém-se a realização de estado discreta para a planta livre $P = (X_p,\ f_p,\ h_p,\ x_p)$. O programa retorna as funções de transição e saída evento da planta livre, bem como uma tabela de tradução, sendo possível identificar os conjuntos de estados, de entradas condição e de saídas evento.

Nesta modelagem, os estados e os sinais são modelados como vetores, onde cada elemento do vetor está relacionado com um elemento do sistema.

O conjunto de estados da planta livre é dado por $X_p = X_{est} \times X_{mesa} \times X_{fur} \times X_{rob}$, conforme mos-

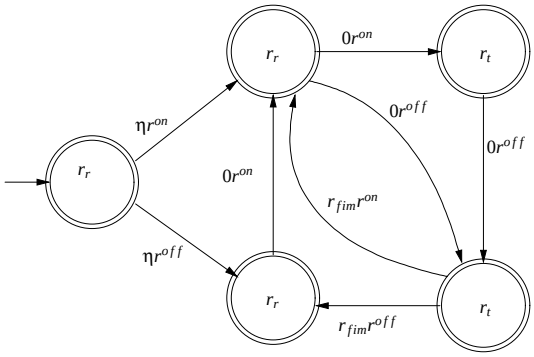


Figura 2.10: Autômato para o manipulador S_{rob}

trado abaixo.

$$X_p = \left\{ \begin{array}{cccc} [& e_m & m_g & f_f & r_t &] \\ [& e_m & m_g & f_f & r_r &] \\ [& e_m & m_g & f_r & r_t &] \\ [& e_m & m_g & f_r & r_r &] \\ [& e_m & m_p & f_f & r_t &] \\ [& e_m & m_p & f_f & r_r &] \\ [& e_m & m_p & f_r & r_t &] \\ [& e_m & m_p & f_r & r_r &] \\ [& e_p & m_g & f_f & r_t &] \\ [& e_p & m_g & f_f & r_r &] \\ [& e_p & m_g & f_r & r_t &] \\ [& e_p & m_g & f_r & r_r &] \\ [& e_p & m_p & f_f & r_t &] \\ [& e_p & m_p & f_f & r_r &] \\ [& e_p & m_p & f_r & r_t &] \\ [& e_p & m_p & f_r & r_r &] \end{array} \right\} = \left\{ \begin{array}{cc} [& x_1 &] \\ [& x_2 &] \\ [& x_3 &] \\ [& x_4 &] \\ [& x_5 &] \\ [& x_6 &] \\ [& x_7 &] \\ [& x_8 &] \\ [& x_9 &] \\ [& x_{10} &] \\ [& x_{11} &] \\ [& x_{12} &] \\ [& x_{13} &] \\ [& x_{14} &] \\ [& x_{15} &] \\ [& x_{16} &] \end{array} \right\}$$

O conjunto de entradas condição da planta livre é dado por $U_p = U_{est} \times U_{mesa} \times U_{fur} \times U_{rob}$, conforme segue.

$$U_p = \left\{ \begin{array}{cccc} [& e^{on} & m^{on} & f^{on} & r^{on} &] \\ [& e^{on} & m^{on} & f^{on} & r^{off} &] \\ [& e^{on} & m^{on} & f^{off} & r^{on} &] \\ [& e^{on} & m^{on} & f^{off} & r^{off} &] \\ [& e^{on} & m^{off} & f^{on} & r^{on} &] \\ [& e^{on} & m^{off} & f^{on} & r^{off} &] \\ [& e^{on} & m^{off} & f^{off} & r^{on} &] \\ [& e^{on} & m^{off} & f^{off} & r^{off} &] \\ [& e^{off} & m^{on} & f^{on} & r^{on} &] \\ [& e^{off} & m^{on} & f^{on} & r^{off} &] \\ [& e^{off} & m^{on} & f^{off} & r^{on} &] \\ [& e^{off} & m^{on} & f^{off} & r^{off} &] \\ [& e^{off} & m^{off} & f^{on} & r^{on} &] \\ [& e^{off} & m^{off} & f^{on} & r^{off} &] \\ [& e^{off} & m^{off} & f^{off} & r^{on} &] \\ [& e^{off} & m^{off} & f^{off} & r^{off} &] \end{array} \right\} = \left\{ \begin{array}{cc} [& u_1 &] \\ [& u_2 &] \\ [& u_3 &] \\ [& u_4 &] \\ [& u_5 &] \\ [& u_6 &] \\ [& u_7 &] \\ [& u_8 &] \\ [& u_9 &] \\ [& u_{10} &] \\ [& u_{11} &] \\ [& u_{12} &] \\ [& u_{13} &] \\ [& u_{14} &] \\ [& u_{15} &] \\ [& u_{16} &] \end{array} \right\}$$

O conjunto de saídas evento da planta livre, dado por $V_p = V_{est} \times V_{mesa} \times V_{fur} \times V_{rob}$, é como

segue.

$$V_p = \left\{ \begin{array}{[ccccc]} [& v_{p_1} & 0 & 0 & 0 &] \\ [& 0 & v_{120} & 0 & 0 &] \\ [& 0 & 0 & f_{fim} & 0 &] \\ [& 0 & 0 & 0 & r_{fim} &] \end{array} \right\} = \left\{ \begin{array}{[cc]} [& v_1 &] \\ [& v_2 &] \\ [& v_3 &] \\ [& v_4 &] \end{array} \right\}$$

O estado inicial da planta é o vetor $x_p = [e_p \;\; m_p \;\; f_r \;\; r_r]$.

Após obter as funções de transição e saída evento é possível encontrar um autômato que represente a linguagem do sistema. Isto é possível através do algoritmo para encontrar o autômato C/E, que também foi implementado em [6]. Após utilizar o programa encontra-se um autômato com 17 estados e 496 transições para representar o comportamento da planta livre. Devido o seu tamanho excessivo (número de estados e transições) o autômato não será mostrado aqui. Entretanto algumas observações devem ser feitas.

Considera-se que transições de estado no modelo global da planta podem acontecer somente nos instantes de ocorrência de eventos. Desta forma, no modelo que representa o comportamento lógico da planta utiliza-se um evento ($v_0 = [\eta \;\; \eta \;\; \eta \;\; \eta]$) para possibilitar a representação da aplicação de um sinal condição inicial (antes da ocorrência de qualquer evento).

Entretanto, para possibilitar transições de estado em um subsistema isolado aconteçam em instantes de ocorrência de eventos de outros subsistemas, assume-se que transições em cada subsistema possam acontecer sem a ocorrência de um evento deste mesmo subsistema. Por exemplo, a transição $0e^{off}$ e $0e^{on}$ da Figura 2.7. Note que ao compor os vários subsistemas de forma a obter o modelo global da planta, seus sinais são sincronizadas ocorrendo transições somente nos instantes de ocorrência de eventos. Para estar de acordo com a restrição de que transições de estado no modelo global da planta acontecem somente nos instantes de ocorrência de eventos, na composição dos subsistemas é preciso eliminar as transições do tipo $v = [0, 0, 0...0]$ (veja que isso não ocorre naturalmente ao compor).

2.4 Sistema Condição/Evento como Modelo para Sistemas Híbridos

Sistemas híbridos, em geral, são sistemas que englobam dois tipos de subsistemas de dinâmicas diferentes: sistemas com dinâmicas contínuas, contendo variáveis ou sinais que adotam valores sobre um conjunto contínuo (e.g. conjunto dos números reais); e sistemas com dinâmicas discretas, que assumem valores sobre um conjunto discreto, tipicamente finito (e.g. conjunto de símbolos).

Nesta seção apresenta-se a modelagem da planta híbrida, utilizando-se para tal o formalismo de sistemas condição/evento. Na literatura, o assunto abordado pode ser encontrado em [2, 3, 4, 8, 9, 13].

2.4.1 Planta Híbrida em Malha Aberta

Considera-se a classe de sistemas híbridos ilustrada na Figura 2.11. A planta hibrída $\hat{\mathcal{H}}$ é composta pela interconexão de um subsistema de dinâmica contínua, $\mathcal{H}_c$, e outro de dinâmica discreta, $\hat{\mathcal{H}}_d$. Assume-se que existe apenas um subsistema de cada tipo, pois múltiplos subsistemas do mesmo tipo podem ser reduzidos a um único através da utilização de vetores de sinais de entrada e saída.

Os aspectos discretos da planta híbrida são caracterizados por sinais condição e sinais evento [17]. O sinal de entrada para o subsistema contínuo, $\mathcal{H}_c$, é um sinal condição, $u(\cdot)$, constante por partes, contínuo à direita e com limites à esquerda, assumindo valores sobre o conjunto finito de condições U. O espaço de todos os sinais condição $u(\cdot)$ em [0, ∞) é denotado por $\mathcal{U}$.

O sinal de saída do subsistema contínuo é um sinal evento, $v(\cdot)$, um sinal que assume valores não nulos apenas em pontos isolados do tempo. O espaço de todos os sinais evento $v(\cdot)$ em [0, ∞) é denotado por $\mathcal{V}$. Assume-se que os sinais evento v e condição u possuem um número finito de descontinuidades em um intervalo finito.

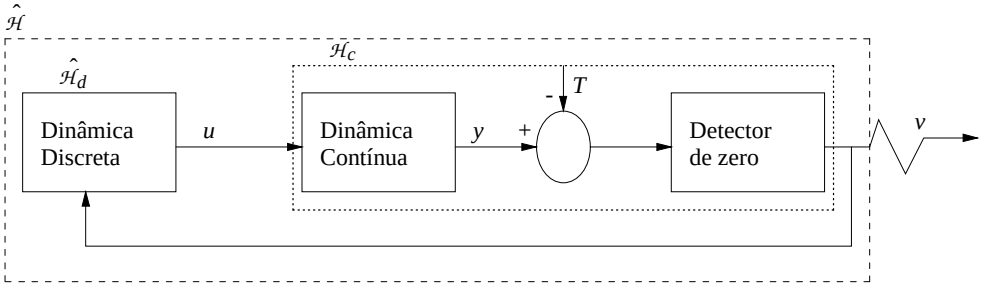


Figura 2.11: Planta hibrída

A seguir descreve-se o comportamento de cada uma das partes da planta híbrida.

Subsistema Contínuo

Conforme ilustrado na Figura 2.11, a parte contínua da planta híbrida é representada pelos dois blocos à direita. A dinâmica contínua da planta é definida pela trajetória de estado contínua $x(\cdot)$ que evolui em $X = R^n$, satisfazendo a equação diferencial $\dot{x} = f_{u(t)}(x(t))$ determinada a cada instante t, pelo sinal condição $u(t)$, onde $f_u : \mathbb{R}^n \to \mathbb{R}^n$ para todo $u \in U$. O valor inicial da trajetória de estados contínua, $x(0)$, pertence ao conjunto de estados iniciais $X_0 \subseteq \mathbb{R}^n$. O conjunto de todas as possíveis trajetórias de estado para um dado sinal de entrada $u(\cdot) \in \mathcal{U}$, iniciando em qualquer estado do conjunto $X' \subset X_0$, é denotado por $\chi_{u(\cdot)}(X')$.

A função $g : X \to R^m$ gera o sinal de saída contínuo, $y(\cdot)$, da trajetória de estados, isto é, $y(\cdot) = g(x(\cdot))$. Cada componente de $y(\cdot)$ é comparada a um patamar, definido pelo vetor de patamares $T \in \mathbb{R}^m$, e o sinal de saída evento é gerado por um detector de zero, definido para cada componente do sinal contínuo de saída $y(\cdot)$, como $v(0) = \eta$ e para $t > 0$ e $i = 1, 2, \ldots, m$ como:

$$v_i(t) = \begin{cases} 1 & se \quad y_i(t) - T_i = 0 \ \bigwedge (\exists \Delta > 0)(\forall \delta \in (0,\Delta)) : y_i(t-\delta) - T_i < 0 \\ 0 & \text{caso contrário} \end{cases} \tag{2.1}$$

Naqueles instantes em que qualquer $v_i(t) = 1$, diz-se que ocorreu um evento de limiar. O evento de inicialização η, ocorre somente no instante $t = 0$, e está associado a escolha não determinística do estado inicial $x(0)$. Assim os sinais eventos $v(\cdot)$ assumem valores sobre o conjunto de eventos de limiares V = $\{0,1\}^m - \{0\}^m$ em pontos isolados do tempo, onde m é a dimensão do sinal de saída contínuo y. Note que o detector de zero é unidirecional, ou seja, um evento de saída só é gerado quando um componente $y_i(t) - T_i$ se aproxima de zero de forma crescente.

Subsistema Discreto

O subsistema discreto $\hat{\mathcal{H}_d}$ é um subsistema de dinâmica puramente discreta que mapeia, de forma não determinística, sinais evento $v(\cdot) \in \mathcal{V}$ em sinais condição $u(\cdot) \in \mathcal{U}$. Assume-se que $\hat{\mathcal{H}_d}$ pode mudar o sinal condição de entrada para $\mathcal{H}_c$ se e somente se um evento de limiar é observado. Portanto, este subsistema possibilita a introdução de informações adicionais sobre restrições nas possibilidades de chaveamento, ou seja, $\hat{\mathcal{H}_d}$ permite modelar restrições sobre as possíveis entradas a serem aplicadas a $\mathcal{H}_c$.

A planta híbrida é modelada utilizando-se o formalismo de sistema condição/evento introduzido em [17]. Antes, porém, define-se o produto cartesiano síncrono de $\mathcal{V} e \mathcal{U}$, denotado por $\mathcal{V} \otimes \mathcal{U}$, como o conjunto de todos os pares $(v(\cdot), u(\cdot)) \in \mathcal{V} \times \mathcal{U}$, tal que descontinuidade em $u(\cdot)$ ocorrem apenas em instantes que $v(\cdot)$ é não nulo.

O subsistema contínuo $\mathcal{H}_c$ é definido como um subconjunto de $\mathcal{V} \otimes \mathcal{U}$, onde $(v(\cdot), u(\cdot)) \in \mathcal{H}_c$ se e somente se existe uma trajetória de estado $x(\cdot) \in \mathcal{X}_{u(\cdot)}(X_0)$ tal que o sinal evento de saída resultante é $v(\cdot)$.

O comportamento discreto do sistema C/E $\mathcal{H}_c$ é descrito pelo seu modelo de estado discreto [17]. Para isto, define-se a representação de traço discreto para $\mathcal{H}_c$ como a 4-tupla (W, β, ρ, W_0), descrita como segue. Considere o sinal condição $w(\cdot)$, constante por partes e contínuo à direita, tomando valores em $W = \mathbb{R}^n$, e com valores iniciais em $W_0 = X_0$. Este sinal registra o valor correspondente da trajetória de estados $x(\cdot)$ apenas nos instantes de descontinuidade em $(v(\cdot), u(\cdot)) \in \mathcal{H}_c$. A função de transição $\beta : W \times U \to W$ para $w(\cdot)$ possui a forma $w(t) = \beta(w(t^-), u(t^-))$ e é tal que:

$$w(t) = \begin{cases} \Phi_{u(t^-)}(t, w(t^-)) & \text{se para algum i = 1,2,.., m, } g_i(\Phi_{(t^-)}(t, w(t^-))) - T_i = 0 \\ & \bigwedge (\exists \Delta > 0)(\forall \delta \in (0,\Delta)) : g_i(\Phi_{(t^-)}(t-\delta, w(t^-))) - T_i < 0 \\ w(t^-) & \text{caso contrário} \end{cases} \tag{2.2}$$

onde $\Phi_u(t, x(t_0))$ é a solução da equação diferencial $\dot{x} = f_u(x)$ para $u \in U$, $t \geqslant t_0$ e valor inicial $x(t_0)$. Utiliza-se a notação t^- para simbolizar o limite pela esquerda ao instante t. A função de saída evento $\rho : W \times W \to V$ é definida como

$$v(t) = \rho(w(t^-), w(t)) \tag{2.3}$$

Note que a saída gera um evento de limiar apenas nos instantes de transição do estado $w(\cdot)$, e que é nula em qualquer outro instante. Este modelo de traço discreto é similar ao modelo de estado discreto apresentado em [17], exceto pela existência de um conjunto de estados infinito e não enumerável.

De forma análoga, o subsistema discreto $\hat{\mathcal{H}_d}$ é definido como $\hat{\mathcal{H}_d} \subseteq \mathcal{V} \otimes \mathcal{U}$, e seu modelo de estado discreto é dado pela 4-tupla $(Q, \hat{\delta}, \phi, q_0)$, onde Q é o conjunto discreto de estados, enumerável e possivelmente infinito, $q_0 = q(0^-)$ é o estado inicial, e

$$\begin{aligned} q(t) &\in \hat{\delta}(q(t^-), v(t)) \\ u(t) &= \phi(q(t)) \end{aligned} \tag{2.4}$$

Por fim, a planta híbrida é um sistema condição/evento $\hat{\mathcal{H}} \subseteq \mathcal{V} \otimes \mathcal{U}$ e seu modelo de estado discreto é obtido pela conexão cascata e realimentação [17] de $\hat{\mathcal{H}_d}$ e $\mathcal{H}_c$.

A seguir são apresentados dois exemplos de modelagem de sistemas híbridos na forma apresentada nesta seção. Antes de apresentá-los, porém, vale ressaltar alguns pontos: *(i)* quando a planta não possui nenhuma restrição de controle, $\hat{\mathcal{H}_d}$ é utilizado apenas como um artifício para possibilitar a modelagem do comportamento da planta livre, incluindo todas as ações de controle; *(ii)* A dinâmica discreta é expressa por autômatos de Moore, sendo que as transições de estados representam os eventos $v \in V$ e as saídas dos estados são as condições $u \in U$.

No primeiro exemplo apresentado a seguir, a planta possui restrições de controle, portanto para alguns eventos em sua entrada, o subsistema discreto pode ter como saída apenas alguns dos sinais condição. Neste caso, $\hat{\mathcal{H}_d}$ permite modelar restrições sobre as possíveis entradas a serem aplicadas a $\mathcal{H}_c$. No segundo exemplo a planta não possui restrições de controle. Então, para cada sinal evento em sua entrada, $\hat{\mathcal{H}_d}$ tem como saída todos os possíveis sinais condição, sem restrições.

Exemplo 2.1 (Sistema de trens)

Este exemplo tem como finalidade ilustrar a classe de sistemas híbridos em questão e foi apresentado em [8, 9]. O sistema consiste de dois trens sobre trilhos cíclicos que compartilham um trecho, conforme ilustrado na Figura 2.12. Os trens podem trafegar em dois modos de velocidade: rápido e devagar, os quais denotaremos por **f** e **s**, respectivamente.

Sobre os trilhos existem sensores que detectam o cruzamento dos trens por determinados pontos: A, B, C e D referente ao trem 1; similarmente E, F e G ao trem 2.

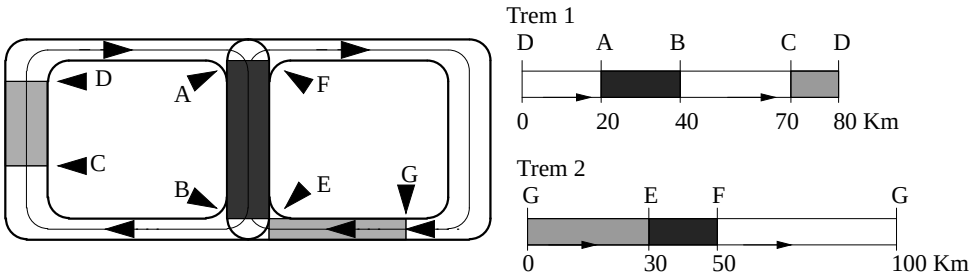


Figura 2.12: Exemplo de sistema de trens

Neste sistema, a velocidade lenta só pode ser adotada entre os trechos C-D para o trem 1, e G-E para o trem 2, trechos estes que são indicados na Figura 2.12 pelo sombreamento claro. Esta limitação implica em uma restrição na possibilidade de chaveamento. O problema de controle consiste em garantir a exclusão mútua dos trens no trecho compartilhado, isto é, no trecho A-B/E-F.

O comprimento de cada um dos trilhos, bem como a disposição dos sensores sobre os mesmos é conforme mostrado no lado direito da Figura 2.12 . Os pontos D e G foram escolhidos como referência para a medida do comprimento dos respectivos trilhos.

A seguir apresenta-se a modelagem deste sistema, sendo esta bastante semelhante àquela detalhada em [9]. O espaço de estados contínuo é dado por $x = [x_1 \quad x_2]^T$, onde x_i é a posição do trem i $(i = 1,2)$, medida a partir da origem. Com o intuito de não introduzir saltos na trajetória de estados contínuos, modelou-se a variável posição de forma que ela assume valores num intervalo $[0, max_i/2]$, onde max_i é o valor da distância máxima do trem i em relação à origem e equivale ao comprimento do trilho em que trafega o trem i. Assim, a posição cresce de 0 até $max_i/2$ e decresce de $max_i/2$ até 0, conforme ilustrado na Figura 2.13.

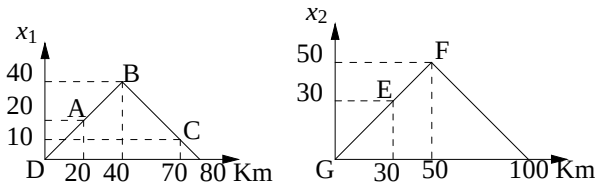


Figura 2.13: Gráfico da posição x_i em função da localização do trem i no trilho correspondente

Sejam x_1 e x_2 as posições dos trens 1 e 2 medidas a partir da origem, respectivamente. Associa-se as dinâmicas possíveis para x_1 e x_2 pelos sinais condição u_1 e u_2 que adotam valores em $U_1 = U_2 = \{1,2,3,4\}$ conforme apresentado na equação (2.5), onde i = 1, 2:

$$\dot{x_i} = \begin{cases} -s & u_i = 1 \\ -f & u_i = 2 \\ s & u_i = 3 \\ f & u_i = 4 \end{cases} \tag{2.5}$$

A trajetória de estados em tempo contínuo $x(\cdot)$ é um vetor coluna 2×1 que armazena os valores de $x_1(\cdot)$ e $x_2(\cdot)$, isto é, $x = [x_1 \;\; x_2]^T$. Toma valores em $([D,B] \times [G,F])^T$, e indexam-se as possíveis dinâmicas para o sinal $x(\cdot)$ pelas condições $u \in U = \{1,...,16\}$, para $t \in [0,\infty)$, como na Tabela 2.9

Tabela 2.9: Dinâmicas para os trens

$\dot{x}$	u_1	u_2	u
$[-s \;\; -s]^T$	1	1	1
$[-s \;\; -f]^T$	1	2	2
$[-s \;\; s]^T$	1	3	3
$[-s \;\; f]^T$	1	4	4
$[-f \;\; -s]^T$	2	1	5
.	.	.	.
.	.	.	.
$[f \;\; f]^T$	4	4	16

O cruzamento de um trem por um sensor é interpretado como a ocorrência de um evento de limiar. Como neste exemplo existem 7 sensores, então $y, T \in \mathbb{R}^7$, como apresentado na equação (2.6). Portanto $v \in \{0,1\}^7 - \{0\}^7$. Por simplicidade, assume-se somente a passagem de um dos trens pelos sensores a cada instante, com isto, existem 7 possíveis eventos de limiar. No domínio de sistemas a eventos discretos utiliza-se símbolos para representar os eventos v. Por exemplo, o símbolo A, é usado para representar o evento $v(t) = [1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0]^T$, o qual representa o cruzamento do trem 1 pela posição A. Assim, os eventos de limiar tomam valores em V = $\{A,B,C,D,E,F,G\}$. Note que somente no instante da ocorrência de um destes eventos é que pode haver um comando de mudança da velocidade de um ou de ambos os trens, sendo que se considera instantânea a dinâmica de mudança de velocidade.

A saída $y(t)$ é comparada com o vetor de patamares T para gerar os eventos de cruzamento $v(t)$. A função $y = g(x)$, o vetor de patamares T e o vetor resultante da operação $y - T$ são apresentados na equação 2.6.

$$y = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ -1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} x \qquad T = \begin{bmatrix} 20 \\ 40 \\ -10 \\ 0 \\ 30 \\ 50 \\ 0 \end{bmatrix} \qquad y - T = \begin{bmatrix} x_1 - 20 \\ x_1 - 40 \\ 10 - x_1 \\ -x_1 \\ x_2 - 30 \\ x_2 - 50 \\ -x_2 \end{bmatrix} \tag{2.6}$$

A dinâmica discreta dos trens é expressa pelos autômatos de Moore mostrados na Figura 2.14, nos quais as saídas dos estados são as condições correspondentes da equação (2.5).

Exemplo 2.2 (Sistema de Nível de Líquido)

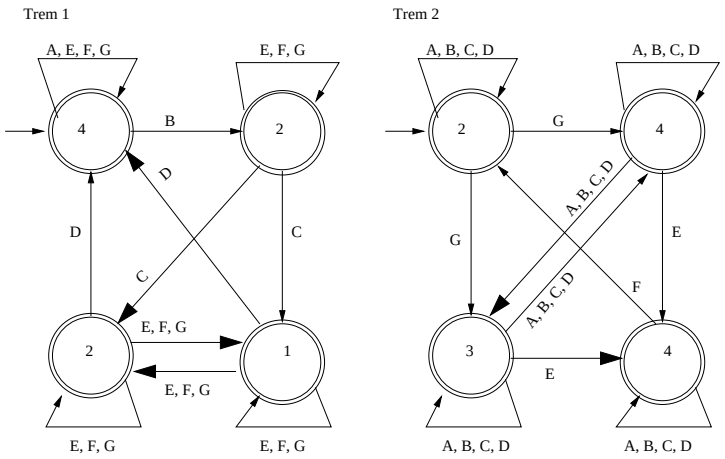


Figura 2.14: Dinâmica discreta para os trens

Considere o sistema de nível de líquido mostrado na Figura 2.15. Para controlar o nível de líquido no tanque pode-se atuar sobre uma válvula que regula a entrada de líquido no mesmo. Para isto, pode-se abrir completamente a válvula, através de um sinal de controle u_{on}, ou então fechar a válvula através do sinal de controle u_{off}. Assim, a variável de controle $u(t)$ é um sinal condição que assume valores sobre o conjunto discreto de condição $U = \{u_{on}, u_{off}\}$.

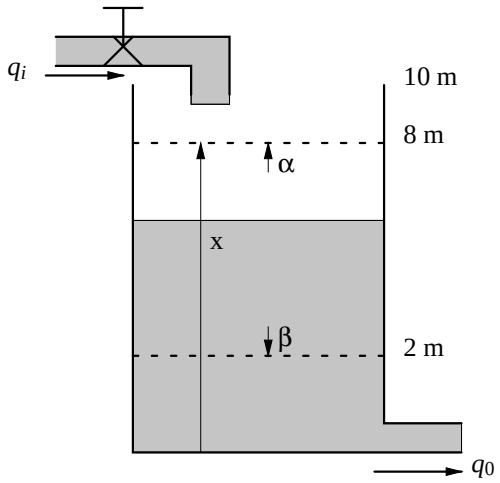


Figura 2.15: Sistema de nível de líquido

A dinâmica contínua da planta pode ser expressa pela seguinte equação:

$$\dot{x} = \begin{cases} K_1 q_i - K_2 x & se \quad u = u_{on} \;\; e \;\; 0 < x < 10 \\ -K_2 x & se \quad u = u_{off} \;\; e \;\; 0 < x < 10 \\ 0 & se \quad u = u_{on} \;\; e \;\; x = 10 \;\; ou \;\; se \;\; u = u_{off} \;\; e \;\; x = 0 \end{cases}$$

Assume-se que para $0 < x < 10$ m a vazão de saída é constante. Assume-se também que a vazão

de entrada é tal que sempre que a válvula de entrada é aberta ocorre o enchimento do tanque e sempre que a válvula de entrada é fechada ocorre o seu esvaziamento. Considera-se que inicialmente o tanque está vazio e que a válvula de entrada está aberta, isto é, $x(0) = 0$ e $u(0) = u_{on}$. A função $y = g(x)$ é dada por $y = [1 \quad -1]^T x$ e o vetor de patamares é dado por $T = [8 \quad -2]^T$. Portanto, tem-se que $y - T = [(x-8) \;\; (2-x)]^T$.

O sinal de saída evento $v(t) = [v_1(t) \;\; v_2(t)]$ toma valores no conjunto $V = \{\alpha, \beta\}$. O evento α é gerado durante o enchimento do tanque, quando o nível atinge 8 metros e ocorre uma descontinuidade em $v_1(t)$. Já o evento β é gerado durante o esvaziamento do tanque, quando o nível atinge o patamar de 2 metros e ocorre uma descontinuidade em $v_2(t)$. Com isto, $\alpha = [1 \;\; 0]$ e $\beta = [0 \;\; 1]$. Observe que quando o nível atinge 10 m e a válvula continua aberta o tanque transborda não havendo mais variação do nível, isto é, $\dot{x} = 0$, e então nenhum evento pode ocorrer. O mesmo acontece quando o tanque se esvazia é a válvula é mantida fechada.

Por fim, a dinâmica discreta da planta é expressa pelo autômato de Moore mostrado na Figura 2.16.

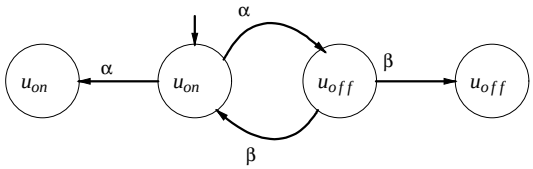


Figura 2.16: Dinâmica discreta para o sistema de nível de líquido

2.5 Controle Supervisório de Sistemas C/E

Esta seção apresenta a teoria de controle supervisório para sistemas C/E e baseia-se nos resultados obtidos em [2, 9, 11]. Provas matemáticas não são apresentadas aqui, mas podem ser encontradas em [11].

2.5.1 Noções Preliminares

Sem perda de generalidade, podemos considerar uma estrutura de controle que se utiliza de um modelo simplificado, conforme mostrado na Figura 2.17. Nesta estrutura, a planta $\mathcal{P}$ é um sistema condição/evento com entrada condição $u(\cdot) \in \mathcal{U}$ e saída evento $v(\cdot) \in \mathcal{V}$ tal que para cada $u(\cdot) \in \mathcal{U}$ exista pelo menos uma saída $v(\cdot) \in \mathcal{V}$ tal que $v(\cdot) \in \mathcal{P}(u(\cdot))$. O supervisor, por sua vez, é um sistema C/E $\mathcal{S}$ definido pelo mapa $\mathcal{S} : \mathcal{V} \to \mathcal{U}$, tal que para cada entrada $v(\cdot) \in \mathcal{V}$ existe pelo menos uma saída $u(\cdot) \in \mathcal{U}$, tal que $u(\cdot) \in \mathcal{S}(v(\cdot))$.

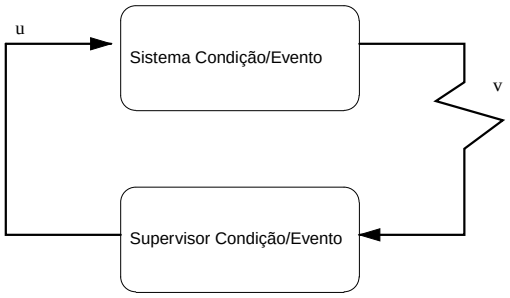


Figura 2.17: Estrutura de Controle

Note então que, a planta possui entrada condição e saída evento, e o supervisor possui entrada evento e saída condição. Nesta estrutura, sinais eventos (transições de estado) na planta forçam transições simultâneas no supervisor e sinais condição do supervisor habilitam ou desabilitam transições de estado na planta. Sendo assim, sinais eventos *v*, fluem da planta para o supervisor, enquanto que sinais condição *u*, fluem do supervisor para a planta. O sinal evento *v*, assume valores sobre um conjunto discreto de eventos V, e o sinal condição *u* assume valores sobre um conjunto discreto de condições U. Sendo ambos os conjuntos não vazios e finitos. Observe que os sinas condição são atualizados somente nos instantes de ocorrência de eventos.

A planta possui uma realização de estado discreta $P = (X, f, h, x_0)$, com os conjuntos U e V (entradas condição e saídas evento), conforme definido anteriormente. O supervisor pode ser definido como uma realização de estado discreta $S = (Y, \xi, \psi, y_0)$ com os conjuntos V e U (os mesmos da planta a ser controlada), onde:

- Y é o conjunto de estados;
- ξ é a função de transição $\xi : Y \times V^+ \rightarrow 2^Y$;
- ψ é a função saída condição $\psi : Y \rightarrow U$;
- y_0 é o estado inicial;
- V é o conjunto de entradas evento;
- U é o conjunto de saídas condição.

O sistema em malha fechada é um sistema C/E $\mathcal{S}/\mathcal{P} \subseteq \mathcal{V} \times \mathcal{U}$ e o seu modelo de estado discreto é obtido pela conexão cascata e realimentação dos modelos do supervisor $\mathcal{S}$ e da planta $\mathcal{P}$ conforme [6, 17].

Percebe-se então que, para descrever os comportamentos lógicos dos sistemas C/E são utilizadas linguagens em $(V^+ \times U)^*$, já para expressar o comportamento desejado para o sistema sob supervisão utiliza-se linguagens em V^*, ou seja, em termos de seqüências de eventos gerados pela planta. Assim,

para que se possa relacionar o comportamento real com o comportamento desejado para o sistema em malha fechada (especificação), define-se a projeção $P_V : (V^+ \times U)^* \to V^*$ como segue.

Definição 2.9 (Projeção P_V:) *A projeção $P_V : (V^+ \times U)^* \to V^*$, é definida recursivamente como:*

$$\begin{array}{lcl} P_V(\varepsilon) & = & \varepsilon; \\ P_V(\sigma) & = & \begin{cases} \varepsilon & \text{para} \quad \sigma = \eta u \in (\{\eta\} \times U) \\ v & \text{para} \quad \sigma = vu \in (V \times U); \quad e \end{cases} \\ P_V(s \circ \sigma) & = & P_V(s) \circ P_V(\sigma) \quad \text{para} \quad s \in (V^+ \times U)^* \; e \;\; \sigma \in (V^+ \times U). \end{array} \tag{2.7}$$

A projeção definida anteriormente pode ser estendida de forma natural para linguagens. Assim, a projeção de uma linguagem $K \subseteq (V^+ \times U)^*$ em V^* é definida como:

$$P_V(K) = \{t \in V^* : \; (\exists s \in K)\, P_V(s) = t\}. \tag{2.8}$$

Desta forma, a projeção $P_V : (V^+ \times U)^* \to V^*$ apaga os símbolos η e os símbolos $u \in U$ de palavras em $(V^+ \times U)^*$.

A projeção inversa de uma linguagem também é definida.

Definição 2.10 (Projeção inversa P_V^{-1} de uma linguagem) *A projeção inversa $P_V^{-1} : V^* \to (\{\eta\} \times U)(V \times U)^*$ de uma linguagem E é definida como:*

$$P_V^{-1}(E) = \{s \in ((\{\eta\} \times U)(V \times U)^*) : P_V(s) \in E\}$$

Desta forma, pode-se apresentar o problema de síntese de supervisores para sistemas C/E, como segue.

Problema 2.1 (Síntese de Supervisores para Sistemas C/E (SSCE)) *Dado o modelo C/E da planta $\mathcal{P} : \mathcal{U} \to \mathcal{V}$ e as especificações $A, E \subseteq V^*$, encontrar um supervisor $\mathcal{S} : \mathcal{V} \to \mathcal{U}$ tal que*

$$A \subseteq P_V[\mathcal{L}_m(\mathcal{S}/\mathcal{P})] \subseteq E.$$

A seguir, utiliza-se a teoria de controle supervisório de SEDs de modo a propor uma solução formal para o problema supra apresentado, como proposto em [9].

2.5.2 Abordagem por Controle de Sistemas a Eventos Discretos

Conforme mostrado em [11], o problema de síntese de supervisores para sistemas C/E pode ser traduzido para uma abordagem de controle puramente discreta. Antes porém, apresenta-se algumas operações básicas sobre linguagens de sistemas C/E, operações estas similares às utilizadas na teoria de SEDs.

Seja uma linguagem $K \subseteq (V^+ \times U)^*$, então o prefixo-fechamento de K , denotado por $\overline{K}$, é definido por:

$$\overline{K} = \{s \in (V^+ \times U)^* : (\exists w \in (V^+ \times U)^*)\ s \circ w \in K\}$$

Em palavras, $\overline{K}$ consiste de todas as cadeias $s \in (V^+ \times U)^*$ que são prefixos de K. Em geral, K $\subseteq \overline{K}$. K é dita prefixo-fechada se $K = \overline{K}$ e é dita ser L_m - fechada se $K = \overline{K} \cap L_m$.

Seja agora uma linguagem $E \subseteq V^*$, então o prefixo-fechamento de E, denotado por $\overline{E}$, é definido por:

$$\overline{E} = \{r \in V^* : (\exists t \in V^*)\ r \circ t \in E\}$$

E é dita ser prefixo-fechada se $E = \overline{E}$ e é dita ser $P_V(L_m)$-fechada se $E = \overline{E} \cap P_V(L_m)$.

O modelo SED com estrutura de controle $P = (L, L_m, \Gamma)$, definido a seguir, possui o mesmo comportamento lógico que o sistema C/E que representa a planta a ser controlada, sendo que $L = \mathcal{L}(\mathcal{P})$ e $L_m = \mathcal{L}_m(\mathcal{P})$.

Definição 2.11 *O Modelo SED com estrutura de controle é a tripla $P = (L, L_m, \Gamma)$, onde $L_m \subseteq L \subseteq (V^+ \times U)^*$ são, respectivamente, a linguagem marcada e gerada; e a estrutura de controle é um mapa $\Gamma : L \to 2^{2^{V^+ \times U}}$ tal que para todo $w \in L$ tem-se que $\Gamma(w) = \{\gamma \in 2^{V^+ \times U} : (\forall v \in V_L(w))(\exists u \in U_L(w, v)) : vu \in \gamma\}$.*

Para compreender a estrutura de controle é necessário definir alguns conjuntos:

- $V_L(w)$ é o conjunto ativo de eventos em L após a cadeia $w \in \overline{L}$, e é definido por $V_L(w) = \{v \in V^+ : (\exists u \in U) w \circ vu \in \overline{L}\}$;
- $U_L(w, v)$ é o conjunto ativo de condições em L após a cadeia $w \in \overline{L}$ para um dado evento $v \in V_L(w)$ e é definido como $U_L(w, v) = \{u \in U : w \circ vu \in \overline{L}\}$.

A estrutura de controle depende da palavra gerada pelo sistema e traduz a idéia de que para um dado evento v, ativo após uma cadeia $w \in \overline{L}$, deve sempre haver pelo menos uma condição u habilitada. Repare que a inibição de todas as condições possíveis para um dado evento levaria a uma situação sem contrapartida física, uma vez que o supervisor inibiria todas as condições aplicáveis em resposta a um dado evento.

Seja o autômato mostrado na Figura 2.18, o qual reconhece a linguagem $L \subseteq (V^+ \times U)^*$. Para $w = \eta u_1 \circ v_1 u_2 \in L$ tem-se que $\Gamma(w) = \{\{v_2u_2, v_3u_1\}, \{v_2u_1, v_3u_1\}, \{v_2u_2, v_2u_1, v_3u_1\}\} = \{\gamma_1, \gamma_2, \gamma_3\}$.

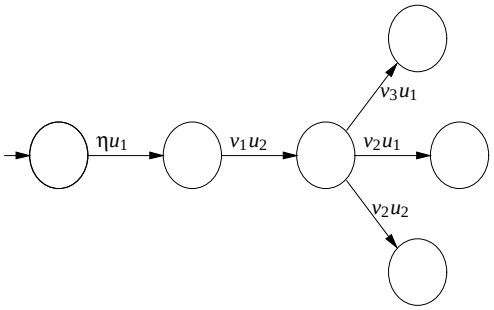


Figura 2.18: Autômato para ilustração de Γ

Observe que $\gamma = \{v_3u_1\} \notin \Gamma(w)$, pois não contempla o evento $v_2 \in V_L(w)$.

Um supervisor S para o sistema $P = (L, L_m, \Gamma)$ é definido pelo mapa $S : L \to 2^{V^+ \times U}$. Um supervisor C/E equivalente ao S pode ser definido como um sistema C/E $\mathcal{S} : \mathcal{L} \to \mathcal{V} \times \mathcal{U}$, tal que:

$$(\forall w \in \mathcal{L}(\mathcal{S}))(\forall vu \in (V^+ \times U))\, w \circ vu \in \mathcal{L}(\mathcal{S}) \iff vu \in S(w).$$

Ou seja, existe equivalência na ação de controle.

A linguagem $L(S/P) \subseteq L$ representa o comportamento da planta $P = (L, L_m, \Gamma)$ sob a ação do supervisor S , e é definida recursivamente como:

1. $\varepsilon \in L(S/P)$; e
2. $w \circ vu \in L(S/P) \iff w \in L(S/P) \wedge w \circ vu \in L \wedge vu \in S(w)$.

O comportamento marcado de S/P é dado por $L_m(S/P) = L(S/P) \cap L_m$, e representa a parte da linguagem marcada da planta que sobrevive à ação de controle.

Definição 2.12 *Um supervisor S é dito ser consistente para uma planta $P = (L, L_m, \Gamma)$ se:*

$$(\forall w \in L(S/P)) \quad S(w) \in \Gamma(w).$$

Deste modo, um supervisor é consistente para uma planta se para qualquer cadeia $w \in L(S/P)$, e para todo evento v factível de acontecer na planta após w, existe pelo menos uma condição de entrada habilitada pelo supervisor, tal que essa entrada pertença ao conjunto de condições válidas na planta.

Neste momento é possível enunciar a seguinte proposição, baseada em [9] e adaptada em [11] para tratar com linguagens marcadas.

Proposição 2.1 *Para o supervisor S para P e um supervisor equivalente $\mathcal{S}$ para $\mathcal{P}$, tem-se que $L(S/P) = \mathcal{L}(\mathcal{S}/\mathcal{P})$ e $L_m(S/P) = \mathcal{L}_m(\mathcal{S}/\mathcal{P})$.*

A proposição mostra que o problema de síntese de supervisores para sistemas C/E (SSCE) pode ser resolvido por um problema de controle supervisório equivalente no domínio de SEDs. Este problema equivalente é enunciado a seguir:

Problema 2.2 (Problema de Controle Supervisório Equivalente - (PCSE)) *Para o problema SSCE definido para planta $\mathcal{P}$ e especificações $A \subseteq E \subseteq V^*$, define-se o problema de controle supervisório equivalente para o modelo de sistemas a eventos discretos $P = (L, L_m, \Gamma)$ com entrada de controle para a planta C/E, como o de encontrar um supervisor S para P tal que :*

$$A \subseteq P_V[L_m(S/P)] \subseteq E.$$

Com o intuito de obter uma solução formal para este problema, apresenta-se a seguir uma série de resultados da teoria de controle supervisório de SEDs, adaptando-os ao contexto deste trabalho, conforme feito em [11, 12].

Um conceito de fundamental importância para a solução de qualquer problema de controle supervisório é o de controlabilidade de linguagens. A seguir, define-se a noção de controlabilidade de linguagens [11] .

Definição 2.13 *Sejam as linguagens $K \subseteq L \subseteq (V^+ \times U)^*$. K é dita ser vu-controlável em relação a L se:*

$$(\forall w \in \overline{K})\, V_L(w) = V_K(w).$$

Logo, a linguagem K é *vu*-controlável em relação a L se para qualquer cadeia de $\overline{K}$ o conjunto ativo de eventos for o mesmo em ambas as linguagens.

Apresentou-se o conceito de controlabilidade no domínio das linguagens em $(V^+ \times U)^*$, mas pode-se estender este conceito para linguagens em V^*, conforme feito a seguir. Note entretanto que uma vez que a controlalilidade está relacionado com os sinais condição, a controlabilidade de uma

linguagem em V^* está diretamente associada à definição de *vu-* controlabilidade apresentada anteriormente. Desta forma, a análise da controlabilidade realmente ocorre no universo das linguagens em $(V^+ \times U)^*$.

Definição 2.14 *Sejam as linguagens $L \subseteq (V^+ \times U)^*$ e $E \subseteq P_V(L)$. E é dita ser v-controlável em relação a L se $\exists K \subseteq L$ vu- controlável em relação a L tal que $P_V(K) = E$.*

Definição 2.15 *Um supervisor consistente S é dito ser não bloqueante para uma planta $P = (L, L_m, \Gamma)$ se $\overline{L_m(S/P)} = L(S/P)$.*

Note que P é não bloqueante se $\overline{L_m} = L$.

Proposição 2.2 *Se uma planta $P = (L, L_m, \Gamma)$ é não bloqueante, então qualquer supervisor S consistente para P é um supervisor não bloqueante para P.*

É importante relatar que o fato de uma planta ser bloqueante, não necessariamente implica que a planta controlada seja bloqueante. Note ainda que, se um supervisor S é não consistente para uma planta P, então não faz sentido mencionar o não bloqueio. Assim, quando utiliza-se a condição de supervisor não bloqueante no restante deste trabalho, subentende-se a consistência do supervisor.

A proposição seguinte apresenta as condições necessárias e suficientes para a existência de um supervisor que garanta o cumprimento de uma especificação $K \subseteq (V^+ \times U)^*$.

Proposição 2.3 *Sejam $P = (L, L_m, \Gamma)$ e $K \subseteq L_m, K \neq \emptyset$. Existe um supervisor não bloqueante S para P tal que $L_m(S/P) = K$ se e somente se K é L_m-fechada e vu-controlável em relação a L.*

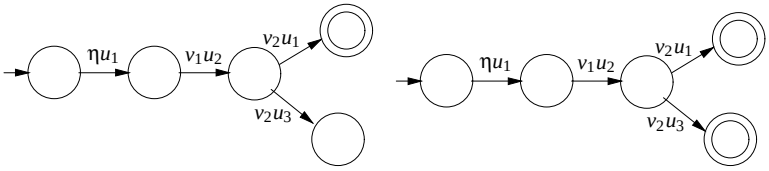
De forma análoga, a proposição seguinte trata da existência de um supervisor que garanta o cumprimento de uma especificação $E \subseteq V^*$. Antes porém apresenta-se alguns conceitos importantes que foram introduzidos em [11].

Definição 2.16 *Seja uma linguagem marcada $K \subseteq (V^+ \times U)^*$. O fechamento para a marcação de K, denotado por $K^{\odot}$, é definido como :*

$$K^{\odot} = \{w \in \overline{K} : (\exists w' \in K)\, P_V(w') = P_V(w)\}$$

Em palavras, se uma cadeia $w = v_1u_1 \circ \cdots \circ v_ku_k$ leva a pelo menos um estado marcado na linguagem K, então todos os estados de K que são alcançados pela seqüência de eventos $v_1 \circ \cdots v_k = P_V(w)$ devem ser marcados, independente das condições u_i associadas aos eventos v_i nestas seqüências. Desta forma, a operação de fechamento para a marcação resume-se em marcar todos os estados alcançados por uma seqüência de eventos que nesta mesma linguagem conduza a pelo menos um estado marcado.

Seja o autômato mostrado na Figura 2.19(a), o qual reconhece a linguagem marcada $K_1 = \{\eta u_1 \circ v_1 u_2 \circ v_2 u_1\}$. A operação de fechamento para a marcação de K_1 consiste em marcar o estado alcançado pela cadeia $w' = \eta u_1 \circ v_1 u_2 \circ v_2 u_3$, conforme ilustrado na Figura 2.19(b)



(a) Autômato que reconhece K_1 (b) Autômato que reconhece $K_1^{\odot}$

Figura 2.19: Autômato para exemplificar a operação $K^{\odot}$

Definição 2.17 *Seja uma linguagem marcada $K \subseteq (V^+ \times U)^*$. K é dita possuir coerência de marcação se $K = K^{\odot}$.*

Assim, uma linguagem possui coerência de marcação se qualquer cadeia w que conduza a um estado marcado, implica em todas as cadeias w' que contêm a mesma seqüência de eventos de w também conduzam a um estado marcado.

A tradução física deste conceito mostra que a realização de tarefas está relacionada a seqüência de eventos, independente das condições de entradas impostas no intuito de garantir a execução da tarefa.

Repare que a coerência de marcação de uma linguagem pode ser enunciada pela seguinte equação:

$$w \in K \Longrightarrow \{w' \in \overline{K} : P_V(w') = P_V(w)\} \subseteq K$$

Proposição 2.4 *Seja uma planta $P = (L, L_m, \Gamma)$, onde L_m possui coerência de marcação, e uma linguagem marcada $E \subseteq V^*, E \neq \emptyset$. Existe um supervisor não bloqueante S para P tal que $P_V[L_m(S/P)] = E$ se e somente se E é $P_V(L_m)$-fechada e v-controlável em relação a L.*

Seja o conjunto de todas as sublinguagens de K que são *vu*-controláveis em relação a L definido como:

$$\mathbb{C}_{VU}(K) = \{K' \subseteq K : (\forall w \in \overline{K'})\ V_{K'}(w) = V_L(w)\}.$$

Pode-se provar que $\mathbb{C}_{VU}(K)$ é não vazio, fechado para a união de conjuntos, e que contém um elemento supremo único, chamado *máxima linguagem vu-controlável*, denotado por $Sup\mathbb{C}_{VU}(K)$. Pode-se provar também que se $K \subseteq (V^+ \times U)^*$ é uma linguagem L_m-fechada, então $Sup\mathbb{C}_{VU}(K)$ também é L_m-fechada. Com base nestas afirmações e na Proposição 2.3, pode apresentar a seguinte proposição.

Proposição 2.5 *Sejam uma planta $P = (L, L_m, \Gamma)$ e $K \subseteq L_m, K \neq \emptyset$. Se $K = \overline{K} \cap L_m$ então existe um supervisor não bloqueante S para P tal que $L_m(S/P) = Sup\mathbb{C}_{VU}(K)$.*

Seja agora o conjunto de todas as sublinguagens de E que são *v*-controláveis em relação a L definido como:

$$\mathbb{C}_V(E) = \{E' \subseteq E : E' \text{ é } v\text{-controlável em relação a } L\}.$$

O conjunto $\mathbb{C}_V(E)$ é não vazio, fechado para união de conjuntos e contém um elemento supremo único, chamado *máxima linguagem v-controlável*, denotada por $Sup\mathbb{C}_V(E)$

Sejam as linguagens $L_m \subseteq L \subseteq (V^+ \times U)^*$ e $E \subseteq V^*$. Se E é $P_V(L_m)$-fechada, então $Sup\mathbb{C}_V(E)$ também é $P_V(L_m)$-fechada. Com base nesta afirmação e na Proposição 2.4 enuncia-se a seguinte proposição.

Proposição 2.6 *Sejam uma planta $P = (L, L_m, \Gamma)$ e uma especificação $E \subseteq V^*, E \neq \emptyset$. Se $E = \overline{E} \cap P_V(L_m)$ então existe um supervisor não bloqueante S para P tal que $P_V[L_m(S/P)] = Sup\mathbb{C}_V(E)$.*

Proposição 2.7 *Sejam as linguagens $L_m \subseteq L \subseteq (V^+ \times U)^*$ e $E \subseteq V^*$. Seja ainda $K = P_V^{-1}(E) \cap L_m$. Se $E \subseteq P_V(L_m)$ então tem -se que $Sup\mathbb{C}_V(E) = P_V[Sup\mathbb{C}_{VU}(K)]$.*

A Proposição 2.7 assegura que a máxima linguagem contida em E que é *v*-controlável em relação a L, é igual à projeção em V^* da máxima linguagem contida em K que é *vu*- controlável em relação a L. Observe que se E consiste na especificação sobre o comportamento lógico do sistema sob supervisão, então K consiste na projeção de E em $(V^+ \times U)^*$ e é encontrada habilitando-se todas as condições factíveis de acontecer na planta para cada cadeia de eventos em E.

Pode-se enunciar então o seguinte teorema.

Teorema 2.3 *O Problema de Controle Supervisório Equivalente para Sistemas C/E (PCSE) possui solução se e somente se $Sup\mathbb{C}_V(E) \supset A$.*

Desta forma um supervisor S para P tal que $P_V[L_m(S/P)] = Sup\mathbb{C}_V(E)$ é uma solução ótima para o PCSE no sentido de ser o menos restritivo possível.

Corolário 2.1 *O problema de Síntese de Supervisores para Sistemas Condição/Evento (SSCE) possui solução se e somente se o Problema de Controle Supervisório Equivalente para Sistemas C/E (PCSE) possui solução.*

Retomando o problema de controle supersisório equivalente para sistemas C/E (PCSE), a teoria apresentada anteriormente demonstra que a solução do PCSE, requer a realização dos seguintes passos:

1. Dada a especificação $E \subseteq V^*$, encontrar a linguagem alvo $K \subseteq L_m \subseteq (V^+ \times U)^*$ tal que $P_V(K) = E$;

2. Verificar se K é *vu*-controlável em relação a L. Caso afirmativo, passar para o passo 3. Caso contrário, deve-se obter a máxima linguagem *vu*-controlável contida em K;

3. O supervisor S é implementado através de um autômato tal que $L_m(S/P) = Sup\mathbb{C}_{VU}(K)$.

Pode-se obter ainda a máxima linguagem *v*-controlável contida em E, fazendo $Sup\mathbb{C}_V(E) = P_V[Sup\mathbb{C}_{VU}(K)]$.

Apresenta-se, a seguir, um exemplo simples e acadêmico com intuito de ilustrar a metodologia proposta. Utiliza-se este por ser reduzido o suficiente de forma a permitir a apresentação detalhada de sua resolução.

2.5.3 Exemplo

Seja um sistema C/E $\mathcal{S}$ cujo o comportamento seqüencial lógico $L_m = \mathcal{L}_m(\mathcal{S})$ é reconhecido pelo autômato ilustrado na Figura 2.20.

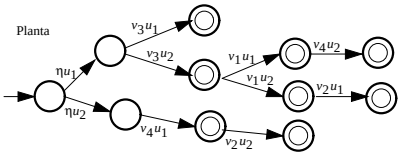


Figura 2.20: Autômato que reconhece a linguagem $L_m = \mathcal{L}_m(\mathcal{S})$

A Figura 2.21 apresenta a especificação $E \subseteq P_V(L_m)$ que modela o comportamento desejado para a planta em malha fechada.

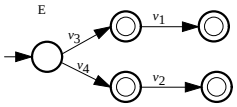


Figura 2.21: Especificação $E \subseteq P_V(L_m)$

O primeiro passo na síntese do supervisor consiste em obter a linguagem alvo $K \subseteq L_m \subseteq (V^+ \times U)^*$. Tal linguagem é obtida fazendo-se $K = P_V^{-1}(E) \cap L_m$ e o autômato que reconhece esta linguagem é ilustrado na Figura 2.22.

Pode-se verificar que a linguagem K não é *vu*-controlável em relação a L. Note, por exemplo, que após a cadeia $w = \eta u_1 \circ v_3 u_2 \circ v_1 u_1 \in \overline{K}$ pode ocorrer o evento v_4 na planta, mas o mesmo não

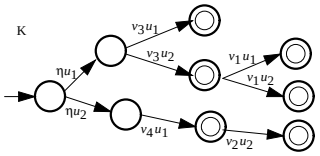


Figura 2.22: Especificação $K \subseteq L_m$

está previsto na especificação, ou seja, $V_L(w) \nsubseteq V_K(w)$. De forma análoga, após a seqüência $w' = \eta u_1 \circ v_3 u_2 \circ v_1 u_2 \in \overline{K}$, a planta prevê a ocorrência do evento v_2, o qual não é previsto na especificação. Encontra-se então a máxima linguagem *vu*-controlável contida em K, denotada por $Sup\mathbb{C}_{VU}(K)$, a qual é reconhecida pelo autômato apresentado na Figura 2.23.

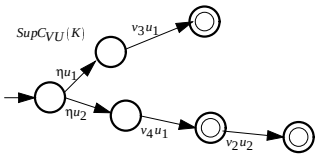


Figura 2.23: Máxima linguagem *vu*-controlável $Sup\mathbb{C}_{VU}(K)$

Vale salientar que $L_m(S/P) = Sup\mathbb{C}_{VU}(K)$. Desta forma, o supervisor S pode ser implementado pelo autômato mostrado na Figura 2.23.

Por fim, pode-se obter a máxima linguagem *v*-controlável contida na especificação $E \subseteq V^*$ fazendo-se $Sup\mathbb{C}_V(E) = P_V[Sup\mathbb{C}_{VU}(K)]$. O autômato que reconhece esta linguagem é ilustrado na Figura 2.24.

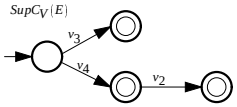


Figura 2.24: Máxima linguagem *v*-controlável $Sup\mathbb{C}_V(E)$

2.6 Conclusão

Neste capítulo apresentou-se os sistemas C/E, uma classe de sistemas a eventos discretos em tempo contínuo que se adapta perfeitamente tanto para à modelagem de sistemas a eventos discretos (SEDs), como para à modelagem de sistemas híbridos (SHs).

Apresentou-se, então, exemplos de SEDs e de SHs modelados como sistemas C/E. Aspectos sobre a modelagem foram abordados e os modelos das plantas livres (modelo global) destes sistemas foram obtidos.

Na seqüência, apresentou-se o problema de controle supervisório para sistemas C/E e uma solução para o mesmo foi apresentada através do uso da teoria de controle supervisório de SEDs.

Capítulo 3

Controle Modular de Sistemas Condição/Evento

Neste capítulo trata-se da abordagem modular de síntese de supervisores para sistemas C/E. Este capítulo está fortemente baseado nos resultados obtidos em [11, 12, 14]. Provas matemáticas não são apresentadas, mas podem ser encontradas em [11, 14].

3.1 Introdução

Normalmente, o comportamento desejado para a planta sob supervisão pode ser expresso através da combinação de duas ou mais especificações. Na abordagem de controle modular, projeta-se um supervisor para cada especificação e aplica-se estes supervisores combinados de modo a se atender a especificação global no sistema resultante em malha fechada [18]. O projeto de cada um dos supervisores modulares é desenvolvido conforme tratado no Capítulo 2.

Em geral, o emprego da abordagem de controle modular, ao invés da abordagem monolítica, é motivada por dois fatores: maior flexibilidade e menor complexidade computacional. Por exemplo, se uma subtarefa é alterada, só é preciso reprojetar o controlador correspondente ao invés de se refazer todo o projeto do supervisor.

Em contrapartida, como a ação de controle de cada supervisor modular não considera as ações dos demais supervisores, pode ser que dois ou mais supervisores entrem em conflito ao atuarem em conjunto para atingir uma especificação global [5, 19]. As condições para que isso não ocorra foram introduzidas em [11, 13] e são apresentadas na seqüência deste capítulo.

3.2 Definição do Problema

O problema de controle modular para sistemas C/E consiste em desenvolver supervisores $\mathcal{S}_i$, cada um para cumprir com uma especificação particular, e implementar o controle global através da conjunção dos supervisores $\mathcal{S}_i$ desenvolvidos. Note que o projeto individual dos supervisores é desenvolvido conforme tratado no Capítulo 2.

Problema 3.1 *Seja a Planta C/E $\mathcal{P}$ com entrada de controle e dada uma especificação $E \subseteq V^*$ para o comportamento da planta em malha fechada, tal que E possa ser subdividida em duas partes, ou seja, $E = E_1 \cap E_2$. Deseja-se encontrar supervisores condição/evento $\mathcal{S}_i$ consistentes para $\mathcal{P}$ tais que $P_V[\mathcal{L}_m(\mathcal{S}_i/\mathcal{P})] \subseteq E_i$, para $i = 1,2$, e de forma que $\mathcal{S}_1 \wedge \mathcal{S}_2$ seja não bloqueante para $\mathcal{P}$ e que $P_V[\mathcal{L}_m((\mathcal{S}_1 \wedge \mathcal{S}_2)/\mathcal{P})] \subseteq E$.*

Da mesma forma que no Capítulo 2, na próxima seção utiliza-se a teoria de controle supervisório de SEDs para propor uma solução formal para o problema supra apresentado.

3.3 Abordagem por Controle de Sistemas a Eventos Discretos

Segundo [12, 13], diferentemente da teoria clássica de controle modular de SEDs [19], para sistemas C/E não basta que as linguagens implementadas por supervisores modulares sejam não conflitantes para garantir que a ação conjunta das mesmas seja não bloqueante e ótima (menos restritiva). Apresenta-se então, o conceito de interconsistência entre linguagens introduzido em [12, 13].

Considere o esquema de controle modular apresentado na Figura 3.1. Deve-se ressaltar que, por motivos de simplicidade, considera-se o caso em que são utilizados apenas dois supervisores.

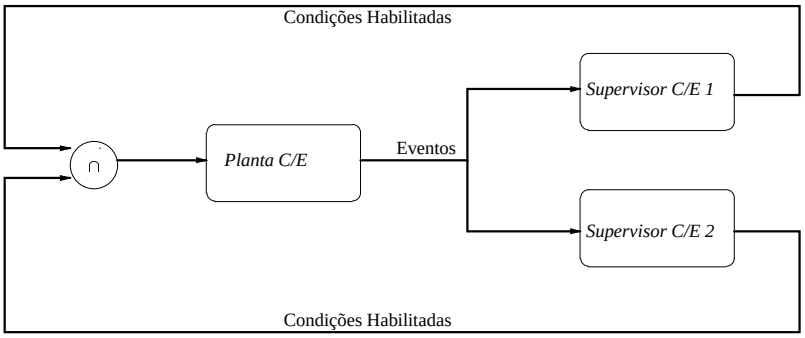


Figura 3.1: Esquema de controle modular para o sistema C/E

Definição 3.1 *Sejam S_1 e S_2 supervisores consistentes para uma planta $P = (L, L_m, \Gamma)$. Define-se a conjunção de S_1 e S_2 como um supervisor dado pelo mapa $(S_1 \wedge S_2) : L \to 2^{V^+ \times U}$ tal que para $w \in L$:*

$$vu \in (S_1 \wedge S_2)(w) \Longleftrightarrow vu \in (S_1(w) \cap S_2(w)).$$

Desta forma, uma par $vu \in V^+ \times U$ é habilitado por $S_1 \wedge S_2$ se e somente se vu é habilitado por S_1 e S_2 simultaneamente. Logo $L((S_1 \wedge S_2)/P) = L(S_1/P) \cap L(S_2/P)$ e $L_m((S_1 \wedge S_2)/P) = L_m(S_1/P) \cap L_m(S_2/P)$.

Proposição 3.1 *Sejam S_1 e S_2 supervisores para P e dados $\mathcal{S}_1$ e $\mathcal{S}_2$ supervisores para $\mathcal{P}$, tais que S_i e $\mathcal{S}_i$ são logicamente equivalentes, $i = 1,2$. Então $S_1 \wedge S_2$ e $\mathcal{S}_1 \wedge \mathcal{S}_2$ são logicamente equivalentes, ou seja:*

$$L((S_1 \wedge S_2)/P) = \mathcal{L}((\mathcal{S}_1 \wedge \mathcal{S}_2)/\mathcal{P}) \text{ e } L_m((S_1 \wedge S_2)/P) = \mathcal{L}_m((\mathcal{S}_1 \wedge \mathcal{S}_2)/\mathcal{P}).$$

Apoiado na Proposição 3.1, pode-se garantir que $S_1 \wedge S_2$ é não bloqueante para P se e somente se $\mathcal{S}_1 \wedge \mathcal{S}_2$ é não bloqueante para $\mathcal{P}$. A Proposição 3.1 propõe que o problema de controle modular de sistemas C/E pode ser resolvido por um problema equivalente no domínio de SEDs.

Definição 3.2 *Duas linguagens $K_1, K_2 \subseteq (V^+ \times U)^*$ são ditas interconsistentes se :*

$$(\forall w \in \overline{K_1} \cap \overline{K_2}) \;\; V_{K_1}(w) \cap V_{K_2}(w) = V_{K_1 \cap K_2}(w).$$

Em palavras, duas linguagens, K_1 e K_2, são interconsistentes se após qualquer cadeia comum a $\overline{K_1}$ e $\overline{K_2}$, os eventos possíveis de ocorrer em ambas linguagens, são também possíveis de ocorrer na linguagem $K_1 \cap K_2$.

Proposição 3.2 *Sejam duas linguagens $K_1, K_2 \subseteq (V^+ \times U)^*$. Se K_1 e K_2 são interconsistentes, então K_1 e K_2 são modulares.*

Note ainda que, mesmo que a interconsistência entre linguagens implique na modularidade entre as mesmas, o inverso não é verdade. Isto é, a modularidade entre linguagens não implica, necessariamente, na interconsistência entre estas. Esta propriedade é ilustrada através do exemplo mostrado a seguir.

Exemplo 3.1 *Seja o autômato ilustrado na Figura 3.2, onde $L_m \subseteq L \subseteq (V^+ \times U)^*$ são, respectivamente, a linguagem marcada e gerada.*

Sejam ainda $K_1, K_2 \subseteq L_m$ duas linguagens vu-controláveis em relação a L. A Figura 3.3 mostra os dois autômatos que reconhecem estas linguagens, respectivamente. Verifica-se facilmente que $\overline{K_1} \cap \overline{K_2} = \overline{K_1 \cap K_2}$ (ver Figura 3.4), de onde conclui-se que as linguagens K_1 e K_2 são modulares (ou não conflitantes). Note agora que para $w = \eta u_1 \circ v_3 u_2 \in \overline{K_1} \cap \overline{K_2}$ tem -se $V_{K_1}(w) \cap V_{K_2}(w) = \{v_2\}$, mas

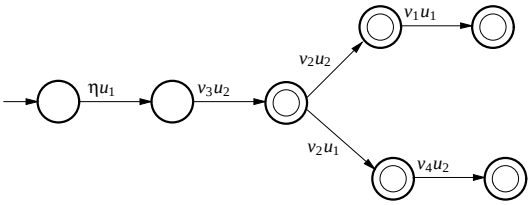
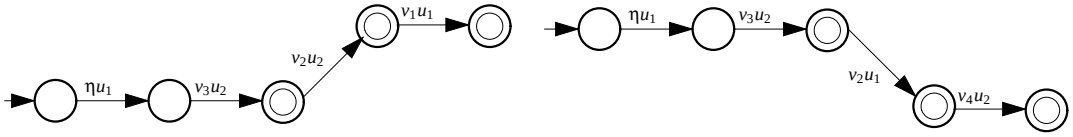


Figura 3.2: Autômato que reconhece as linguagens L e L_m do Exemplo 3.1



(a) Autômato que reconhece K_1.

(b) Autômato que reconhece K_2.

Figura 3.3: Linguagens *vu*-controláveis em relação a L.

que $V_{K_1\cap K_2}(w)=\emptyset$, logo, $\exists w\in\overline{K_1}\cap\overline{K_2}: V_{K_1}(w)\cap V_{K_2}(w)\nsubseteq V_{K_1\cap K_2}(w)$, e portanto K_1 e K_2 não são interconsistentes. Assim, conclui-se que, embora as linguagens K_1 e K_2 sejam modulares, elas não são interconsistentes.

Proposição 3.3 *Dados uma planta $P=(L,L_m,\Gamma)$ e duas especificações $K_1,K_2\subseteq L_m$. Sejam S_1 e S_2 supervisores não bloqueantes para P tais que $L_m(S_i/P)=K_i$, sendo $i=1,2$. O supervisor $S_1\wedge S_2$ é consistente e não bloqueante para P se e somente se K_1 e K_2 são interconsistentes.*

Proposição 3.4 *Dadas $K_1,K_2\subseteq(V^+\times U)^*$. Se $Sup\mathbb{C}_{VU}(K_1)$ e $Sup\mathbb{C}_{VU}(K_2)$ são interconsistentes, então:*

$$Sup\mathbb{C}_{VU}(K_1)\cap Sup\mathbb{C}_{VU}(K_2)=Sup\mathbb{C}_{VU}(K_1\cap K_2).$$

O teorema apresentado a seguir introduz uma condição necessária e suficiente para a solução do Problema de Controle Modular Ótimo para Sistemas C/E.

Teorema 3.1 *Seja uma planta $P=(L,L_m,\Gamma)$, cuja linguagem L_m possui coerência de marcação, e duas especificações, E_1 e E_2, $P_V(L_m)$-fechadas. Existem dois supervisores, S_1 e S_2, tais que:*

1. *S_i é não bloqueante para P e $P_V[L_m(S_i/P)]=Sup\mathbb{C}_V(E_i),(i=1,2)$;*
2. *$S_1\wedge S_2$ é não bloqueante para P; e*

Figura 3.4: Autômato que reconhece a linguagem $\overline{K_1}\cap\overline{K_2}=\overline{K_1\cap K_2}$.

3. $P_V[L_m((S_1 \wedge S_2)/P)] = Sup\mathbb{C}_V(E_1) \cap Sup\mathbb{C}_V(E_2) = Sup\mathbb{C}_V(E_1 \cap E_2)$,

se e somente se $Sup\mathbb{C}_{VU}[P_V^{-1}(E_1) \cap L_m]$ *e* $Sup\mathbb{C}_{VU}[P_V^{-1}(E_2) \cap L_m]$ *são interconsistentes.*

Suponha que uma especificação $E \subseteq V^*$ para o comportamento da planta P em malha fechada é tal que E possa ser subdividida em duas partes, ou seja, $E = E_1 \cap E_2$. Se $Sup\mathbb{C}_{VU}[P_V^{-1}(E_1) \cap L_m]$ e $Sup\mathbb{C}_{VU}[P_V^{-1}(E_2) \cap L_m]$ são interconsistentes, então existem supervisores S_i não bloqueantes para P que implementam as máximas linguagens *v*-controláveis contidas em E_i, e cuja a ação conjunta dos supervisores resulta numa ação de controle não bloqueante e ótima (minimamente restritiva) dada por $P_V[L_m((S_1 \wedge S_2)/P)] = Sup\mathbb{C}_V(E_1) \cap Sup\mathbb{C}_V(E_2) = Sup\mathbb{C}_V(E_1 \cap E_2)$. Isto é, a solução obtida através da abordagem modular é igual àquela obtida utilizando-se a abordagem monolítica.

Corolário 3.1 *Seja* $P = (L, L_m, \Gamma)$ *o modelo de sistemas a eventos discretos com entrada de controle para o sistema C/E* $\mathcal{P}$ *e sejam os supervisores* S_i *conforme o Teorema 3.1, sendo* $i = 1, 2$. *Os supervisores condição/evento* $\mathcal{S}_i$ *equivalentes aos supervisores* S_i *levam a uma solução ótima para o problema de controle modular de sistemas C/E.*

3.3.1 A existência de solução ótima para a abordagem modular

Com os resultados estabelecidos até o momento, conclui-se que, se $Sup\mathbb{C}_{VU}[P_V^{-1}(E_1) \cap L_m]$ e $Sup\mathbb{C}_{VU}[P_V^{-1}(E_2) \cap L_m]$ são interconsistentes, então a abordagem modular leva a uma solução ótima para o problema de controle modular de sistemas C/E. A questão que surge então é a seguinte: O que fazer se estas linguagens não forem interconsistentes? Esta questão foi tratada em [11, 14] e será discutida a seguir.

Seja $\mathrm{I}(K_1, K_2)$ o conjunto de todas as sublinguagens de K_1 que são interconsistentes em relação a K_2. Pode-se provar que $\mathrm{I}(K_1, K_2)$ é não vazio, fechado para união de conjuntos, e que contém um elemento supremo único, chamado "*máxima linguagem interconsistente*". Denota-se este elemento supremo por $Sup\mathrm{I}(K_1, K_2)$. Provas matemáticas relacionadas com esta subseção não são apresentadas, mas podem ser encontradas em [11, 14].

O procedimento para calcular $Sup\mathrm{I}$ consiste basicamente em remover todos os estados (e transições relacionadas com estes estados) no qual o teste de interconsistência falha ("maus estados"). Este procedimento é iterativo, porque a remoção destes estados pode modificar o resultado do teste para os estados analisados em interações precedentes. A seguir apresenta-se um exemplo de forma a ilustrar a necessidade do teste ser feito de forma iterativa. Sejam K_1 e K_2 as linguagens reconhecidas pelos autômatos mostrados na Figura 3.5. Ao testar a interconsistência entre estas linguagens observa-se que o estado 2 é o único "mau estado". Assim, no procedimento de obtenção da máxima linguagem K_1 que é interconsistente com K_2 ele é eliminado, obtendo-se o resultado mostrado na Figura 3.6.

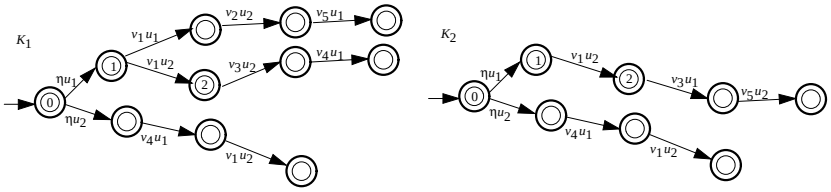


Figura 3.5: Teste da interconsistência

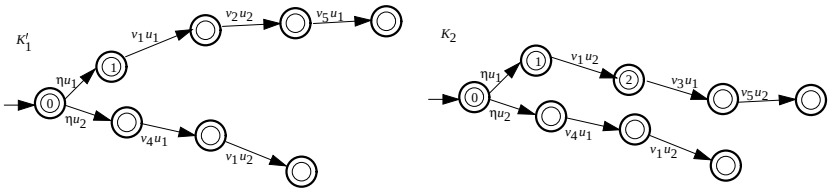


Figura 3.6: K_1' e K_2

Fazendo agora o teste da interconsistencia entre K_1' e K_2 pode-se observar que o estado 1 tornou-se um "mau-estado". Portanto, conclui-se que o teste da interconsistência deve ser feito de forma iterativa.

Seja $\mathrm{I}\mathbb{C}_{VU}(K_1,K_2) = \mathrm{I}(K_1,K_2) \cap \mathbb{C}_{VU}(K_1,K_2)$, o conjunto de todas sublinguagens de K_1 que são *vu*-controláveis em relação a L e interconsistentes em relação a K_2. Como $\mathrm{I}(K_1,K_2)$ e $\mathbb{C}_{VU}(K_1,K_2)$ são não vazios e fechados para união de conjuntos, então conclui-se que $\mathrm{I}\mathbb{C}_{VU}(K_1,K_2)$ é não vazio e fechado para união de conjuntos. Vamos supor que K_1 e K_2 são *vu*-controláveis em relação a L, mas não são interconsistentes. Em geral, $Sup\mathrm{I}(K_1,K_2)$ pode não ser *vu*-controlável em relação a L, isto é, a computação de $Sup\mathrm{I}$ pode destruir a controlabilidade. Supõe agora que K_1 e K_2 são interconsistentes, mas K_1 não é *vu*-controlável em relação a L. É possível que $Sup\mathbb{C}_{VU}(K_1)$ e K_2 não sejam interconsistentes, isto é, a computação de $Sup\mathbb{C}_{VU}$ pode destruir a interconsistência. O procedimento para obter $Sup\mathrm{I}\mathbb{C}_{VU}$ alterna a computação de $Sup\mathbb{C}_{VU}$ e $Sup\mathrm{I}$ até que ambas propriedades (*vu*-controlabilidade e Interconsistência) sejam atingidas. Note que este procedimento converge em um número finito de iterações.

Pode-se, neste momento, enunciar o seguinte teorema.

Teorema 3.2 *Seja $P = (L, L_m, \Gamma)$ e $E_1, E_2 \subseteq P_V(L_m)$. Sejam S_1 e S_2 supervisores , tais que $L_m(S_1/P) = Sup\mathrm{I}\mathbb{C}_{VU}(K_1^{\uparrow}, K_2^{\uparrow})$ e $L_m(S_2/P) = K_2^{\uparrow}$, onde $K_i^{\uparrow} = Sup\mathbb{C}_{VU}[P_V^{-1}(E_i) \cap L_m]$, $i = 1,2$. Então $S_1 \wedge S_2$ é um supervisor consistente para P tal que :*

1. $L_m(S_1 \wedge S_2/P) = (K_1 \cap K_2)^{\uparrow}$ *; e*
2. $P_V[L_m((S_1 \wedge S_2)/P)] = Sup\mathbb{C}_V(E_1 \cap E_2)$.

Proposição 3.5 *Seja $L_m \subseteq L \subseteq (V^+ \times U)^*$ e $E_1, E_2 \subseteq P_V(L_m)$. Seja $K_i^{\uparrow} = Sup\mathbb{C}_{VU}[P_V^{-1}(E_i) \cap L_m]$, onde $i = 1,2$. Então $Sup\mathrm{I}\mathbb{C}_{VU}(K_1^{\uparrow}, K_2^{\uparrow}) \cap K_2^{\uparrow} = Sup\mathrm{I}\mathbb{C}_{VU}(K_1^{\uparrow}, K_2^{\uparrow}) \cap K_1^{\uparrow}$.*

O Teorema 3.2 e a Proposição 3.5 indicam que, se $K_1^{\uparrow}$ e $K_2^{\uparrow}$ não são interconsistentes, então fixa-se uma destas linguagens (por exemplo, $K_2^{\uparrow}$) e obtém-se a máxima sublinguagem *vu*-controlável em relação a L e que é interconsistente em relação a linguagem fixada (neste caso $K_2^{\uparrow}$). Note que a solução ótima é garantida independentemente de qual linguagem ($K_1^{\uparrow}$ ou $K_2^{\uparrow}$) é fixada.

Retornando ao problema de controle modular de sistemas C/E, podemos obter o seguinte resultado.

Corolário 3.2 *Seja $P = (L, L_m, \Gamma)$ o modelo de sistemas a eventos discretos com entrada de controle para o sistema C/E $\mathcal{P}$ e sejam os supervisores S_i conforme o Teorema 3.2, sendo $i = 1, 2$. Os supervisores condição/evento $\mathcal{S}_i$ equivalentes aos supervisores S_i levam a uma solução ótima para o problema de controle modular de sistemas C/E.*

3.3.2 Exemplo

A seguir apresenta-se um exemplo simples com intuito de ilustrar a metodologia de resolução de problemas de controle modular de sistemas C/E utilizando-se a teoria de controle supervisório de SEDs. Este mesmo exemplo foi utilizado no Capítulo 2 (Seção 2.5.3) para ilustrar a utilização da abordagem monolítica e é retomado aqui de forma a permitir uma comparação dos resultados obtidos.

Seja um sistema C/E $\mathcal{S}$ cujo comportamento lógico seqüencial $L_m = \mathcal{L}_m(\mathcal{S})$ é reconhecido pelo autômato ilustrado na Figura 3.7, e sejam as especificações $E_1, E_2 \subseteq P_V(L_m)$ mostradas na Figura 3.8.

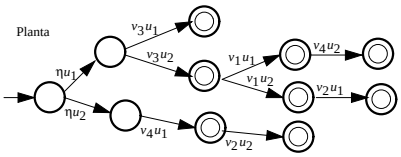


Figura 3.7: Autômato que reconhece a linguagem $L_m = \mathcal{L}_m(\mathcal{S})$

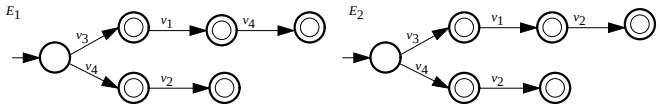


Figura 3.8: Especificações $E_1, E_2 \subseteq P_V(L_m)$

O primeiro passo para a síntese de supervisores consiste em obter especificações equivalentes que estejam contidas na linguagem da planta. Tais especificações são obtidas fazendo-se $K_i = P_V^{-1}(E_i) \cap L_m$, para $i = 1, 2$. Os autômatos que reconhecem estas linguagens são ilustrados na Figura 3.9. Percebe-se que as linguagens K_1 e K_2 não são *vu*-controláveis em relação a L, e então obtém-se as máximas linguagens *vu*-controláveis contidas nestas especificações. Estas são reconhecidas pelos autômatos apresentados na Figura 3.10. Neste instante, testa-se a interconsistencia entre $Sup\mathbb{C}_{VU}(K_1)$

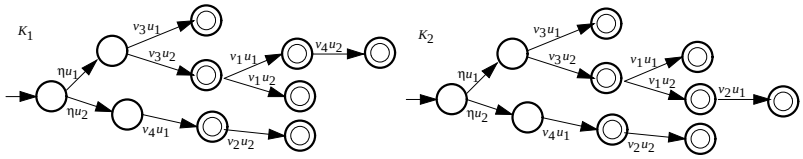


Figura 3.9: Especificações $K_1, K_2 \subseteq L_m$

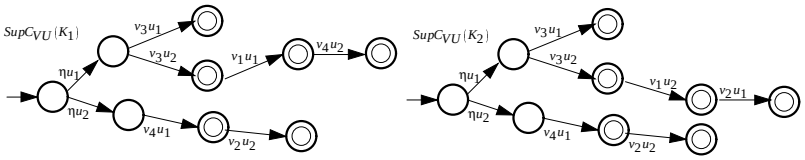
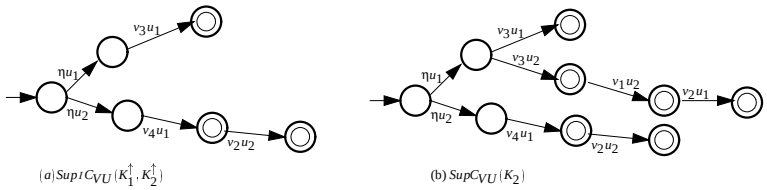


Figura 3.10: Máximas linguagens *vu*-controláveis

e $Sup\mathbb{C}_{VU}(K_2)$. Neste caso tal condição não é verificada. Então o que fazer? A teoria apresentado na subseção 3.3.1 indica uma solução. Fixamos uma destas linguagens (por exemplo, $Sup\mathbb{C}_{VU}(K_2)$) e obtemos a máxima sublinguagem *vu*-controlável em relação a L e que é interconsistente em relação a linguagem fixada. Obtendo assim $Sup\mathrm{I}\mathbb{C}_{VU}(K_1^{\uparrow}, K_2^{\uparrow})$, onde $K_i^{\uparrow} = Sup\mathbb{C}_{VU}(K_i)$, com $i = 1,2$. Desta forma, os supervisores S_1 e S_2 podem ser implementados pelos autômatos mostrados na Figura 3.11(a) e 3.11(b), respectivamente. Pode-se obter ainda a máxima linguagem *v*-controlável contida



(a) $Sup\mathrm{I}C_{VU}(K_1^{\uparrow}, K_2^{\uparrow})$ (b) $SupC_{VU}(K_2)$

Figura 3.11: (a) $Sup\mathrm{I}\mathbb{C}_{VU}(K_1^{\uparrow}, K_2^{\uparrow})$, (b) $Sup\mathbb{C}_{VU}(K_2)$

na especificação $E_i \subseteq V^*$ fazendo-se $Sup\mathbb{C}_V(E_i) = P_V[Sup\mathbb{C}_{VU}(K_i)]$ sendo $i = 1,2$. Os autômatos que reconhecem estas linguagens são ilustrados na Figura 3.12. Por fim, a linguagem resultante da ação conjunta dos dois supervisores, dada por $P_V[L_m((S_1 \wedge S_2)/P)] = Sup\mathbb{C}_V(E_1) \cap Sup\mathbb{C}_V(E_2)$, é mostrada na Figura 3.13. Comparando o resultado obtido na resolução deste problema com o resultado obtido através da abordagem monolítica (Capítulo 2), constata-se que $P_V[L_m((S_1 \wedge S_2)/P)] = Sup\mathbb{C}_V(E_1) \cap Sup\mathbb{C}_V(E_2) = Sup\mathbb{C}_V(E_1 \cap E_2)$, ou seja, a solução obtida através da abordagem modular é igual àquela obtida utilizando-se a abordagem monolítica.

3.4 Conclusão

Na síntese de supervisores modulares, ao invés de projetar um único supervisor monolítico que satisfaça todas as especificações, procura-se construir um supervisor para cada especificação, de forma que atuando em conjunto, satisfaça a especificação global. Sendo assim, a síntese modular permite

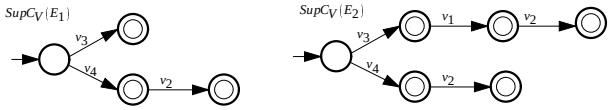


Figura 3.12: Máximas linguagens *v*-controláveis.

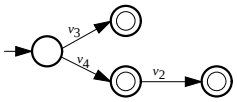


Figura 3.13: $P_V[L_m((S_1 \wedge S_2)/P)] = Sup\mathbb{C}_V(E_1) \cap Sup\mathbb{C}_V(E_2) = Sup\mathbb{C}_V(E_1 \cap E_2)$.

que problemas complexos possam ser decompostos em problemas mais simples, de forma a atribuir maior flexibilidade ao supervisor resultante. Além de ser mais facilmente construído, um supervisor modular costuma ser mais facilmente modificado, atualizado e corrigido. Por exemplo, se uma tarefa é alterada, só é preciso reconstruir o supervisor referente a esta tarefa ao invés de refazer todo o projeto do supervisor.

Mostrou-se que, diferentemente da teoria clássica de controle modular de SEDs [19], para sistemas C/E não basta que as linguagens sejam não conflitantes para garantir que a ação conjunta dos supervisores modulares seja não bloqueante e ótima. Em [11, 14] foi apresentado uma condição sob a qual a solução modular é não bloqueante e ótima e foi proposta uma metodologia para a obtenção desta condição. Assim, o uso desta metodologia garante que a solução obtida através da abordagem modular seja idêntica à solução obtida através da abordagem monolítica.

Capítulo 4

Algoritmos para Controle Supervisório de Sistemas Condição/Evento

É neste capítulo que se concentram as maiores contribuições deste trabalho. Nele serão apresentados algoritmos envolvidos na solução do problema de síntese de supervisores para sistemas C/E sob o ponto de vista da teoria de controle supervisório de SEDs.

4.1 Algoritmos Gerais

O primeiro algoritmo tem como objetivo obter de forma sistemática a projeção de uma linguagem em $(V^+ \times U)^*$ em uma linguagem contida em V^*, conforme apresentado na Definição 2.9 (Seção 2.5). O segundo algoritmo tem como finalidade obter a especificação equivalente $K \subseteq (V^+ \times U)^*$ a partir de uma especificação $E \subseteq V^*$. Estes algoritmos são utilizados na resolução do problema de síntese de supervisores para sistemas C/E, tanto na abordagem modular quanto na monolítica. Deve-se ressaltar que a notação v/u utilizada a seguir é análoga à notação vu utilizada anteriormente neste trabalho e que foi adotada para facilitar o entendimento dos algoritmos.

4.1.1 Algoritmo para obter a projeção $P_V : (V^+ \times U)^* \rightarrow V^*$

Algumas definições são necessárias para o entendimento do Algoritmo 4.1.

- O autômato que reconhece a linguagem K é dado por $G = (X, \Sigma_G, f, x_0, X_m)$, onde X é o conjunto de estados, Σ_G é o alfabeto em $(V^+ \times U)^*$, f é a função de transição, x_0 é o estado inicial e X_m é o conjunto de estados marcados. A notação $f(x, v/u)!$ é utilizada sempre que a função f está definida para o par $(x, v/u)$.

- O modelo que representa a projeção de K é representado pelo Autômato $P = (Q, \Sigma_P, \beta, q_0, Q_m)$, onde Q é o conjunto de estados, Σ_P é o alfabeto em V^*, β é a função de transição, q_0 é o estado inicial e Q_m é o conjunto de estados marcados. Este modelo será obtido no decorrer do algoritmo e determinizado no final.
- X_{alc} é um subconjunto de X. Os estados de X_{alc} são representados por x_{alc}.
- γ é uma variável que armazena as etiquetas de transição no domínio de V.
- $proj$ é um operador que recebe as etiquetas de transição no domínio de $(V^+ \times U)$ e retorna no domínio de V, ou seja, apaga todos os símbolos em U e os símbolos η pertencentes a $V^+ = V \cup \{\eta\}$.
- *aux* é uma variável que recebe estados de X.
- G_{pares} é o conjunto que armazena pares do tipo (q, x), onde $q \in Q$ e $x \in X$.
- *Testa* é o conjunto que armazena pares do tipo (q_a, x_a), no qual $q_a \in Q$ e $x_a \in X$. Com o intuito de diferenciar os conjuntos G_{pares} e *Testa*, referenciamos os estados de Q e X pertencentes ao conjunto *Testa* como q_a e x_a, respectivamente.
- a notação $\leftarrow$ indica atribuição de valores e os textos entre chaves correspondem a comentários.

Observação: Considera-se que as transições relacionadas com o estado inicial de G são do tipo η/u, uma vez que os modelos utilizados para o controle supervisório de sistemas C/E possuem esta característica.

Resumo do algoritmo:

Inicialmente o conjunto de estados Q possui apenas o estado inicial q_0 e o conjunto de estados marcados Q_m é inicializado como vazio. Os conjuntos de estados Q e Q_m são definidos de forma incremental (acumulativa) no decorrer do algoritmo.

O algoritmo proposto obtém todos os estados alcançáveis a partir do estado inicial de G (com etiquetas do tipo η/u), e então associa as transições relacionadas com estes estados ao estado inicial de P (q_0), apagando as condições associadas aos eventos. Para isto, relaciona-se cada estado de Q a um estado de X, com o intuito de obter as transições relacionadas com o estado de Q, apagando todas as condições (u) correspondentes às transições do estado de X, e ainda permite definir quais estados de Q pertencem ao conjunto de estados marcados (Q_m). Por fim, realiza a determinização do autômato P.

4.1.2 Algoritmo para obter a linguagem $K \subseteq (V^+ \times U)^*$

Algumas definições são necessárias para o entendimento do Algoritmo 4.2.

Algoritmo 4.1 Algoritmo para obter a projeção $P_V : (V^+ \times U)^* \rightarrow V^*$

$Q \leftarrow \{q_0\}$; $Q_m \leftarrow \emptyset$; $X_{alc} \leftarrow \emptyset$; {Inicialização dos conjuntos que armazenam estados}
$Testa \leftarrow \emptyset$; $G_{pares} \leftarrow \emptyset$; {Inicialização dos conjuntos que armazenam pares do tipo (q,x) }
para $\forall u \in U$ **faça**
  $X_{alc} \leftarrow f(x_0, \eta/u)$ {Obtém todos os estados alcançáveis a partir do estado inicial de G}
fim para
para $\forall x_{alc} \in X_{alc}$ **faça**
  se $x_{alc} \in X_m$ **então**
    $Q_m \leftarrow q_0$ {Se algum estado alcançável é marcado, então o estado inicial de P, também será marcado $(q_0 \in Q_m)$.}
  fim se
  para $\forall vu \in \Sigma_G$ tal que $f(x_{alc}, v/u)!$ **faça**
    Cria um novo estado q' em Q
    $aux \leftarrow f(x_{alc}, v/u)$
    $\gamma \leftarrow proj(v/u)$
    $\beta(q_0, \gamma) \leftarrow q'$
    $G_{pares} \leftarrow (q', aux)$ {$G_{pares} = G_{pares} \cup (q', aux)$}
    $Testa \leftarrow (q', aux)$ {acrescenta um novo par (q', aux) no conjunto $Testa$}
  fim para
fim para
enquanto $G_{pares} \neq \emptyset$ **faça**
  Selecione e remova um par $(q,x) \in G_{pares}$
  se $x \in X_m$ **então**
    $Q_m \leftarrow q$
  fim se
  para $\forall vu \in \Sigma_G$ tal que $f(x, v/u)!$ **faça**
    $aux \leftarrow f(x, v/u)$
    se $\exists (q_a, x_a) \in Testa$ tal que $(q_a \in Q \wedge x_a = aux)$ **então**
      $\gamma \leftarrow proj(v/u)$
      $\beta(q, \gamma) \leftarrow q_a$
    senão
      Cria um novo estado q' em Q
      $aux \leftarrow f(x_{alc}, v/u)$
      $\beta(q, \gamma) \leftarrow q'$
      $G_{pares} \leftarrow (q', aux)$
      $Testa \leftarrow (q', aux)$
    fim se
  fim para
fim enquanto
Determiniza o autômato P.

- O modelo que representa o comportamento desejado para o sistema em termos de eventos é representado por $E = (X, \Sigma_E, f, x_0, X_m)$, onde X é o conjunto de estados, Σ_E é o alfabeto em V^*, f é a função de transição, x_0 é o estado inicial e X_m é o conjunto de estados marcados. A notação $f(x,v)!$ é utilizada sempre que a função f está definida para o par (x,v).

- O modelo que representa a projeção inversa de E é representado pelo Autômato $I = (Q, \Sigma_I, \lambda, q_0, Q_m)$, onde Q é o conjunto de estados, Σ_I é o alfabeto em $(V^+ \times U)^*$, λ é a função de transição, q_0 é o estado inicial e Q_m é o conjunto de estados marcados. Este modelo será obtido no decorrer do algoritmo apresentado.

- A planta do sistema é modelada como um autômato $P = (Z, \Sigma_Z, \beta, z_0, Z_m)$, onde Z é o conjunto de estados, Σ_P é o alfabeto em $(V^+ \times U)^*$, β é a função de transição, z_0 é o estado inicial e Z_m é o conjunto de estados marcados.

- A linguagem alvo K, também modelada como um autômato, é resultante da intersecção da linguagem reconhecida por P com a linguagem reconhecida por I $(K = P \cap I)$, e será obtida no final da execução do algoritmo.

- γ é uma variável que armazena as etiquetas $vu \in (V^+ \times U)$ referentes às transições de estado, sendo que $V^+ = V \cup \{\eta\}$.

- G_{pares} é o conjunto que armazena pares do tipo (q,x), onde $q \in Q$ e $x \in X$.

- *aux* é uma variável que recebe estados de $x \in X$.

- *Testa* é o conjunto que armazena pares do tipo (q_a, x_a), no qual $q_a \in Q$ e $x_a \in X$.

Resumo do algoritmo:

Inicialmente os conjuntos de estados Q e Q_m são inicializados apenas com o estado inicial (q_0) e são definidos de forma incremental (acumulativa) no decorrer do algoritmo.

O algoritmo obtém todas as transições η/u relacionadas com estado inicial, e em seguida obtém todas as transições v/u relacionadas com os estados de Q. Para isto, relaciona-se cada estado de Q (exceto o estado inicial) com um estado de X com intuito de obter todas as transições v/u associadas ao estado de Q e ainda definir os estados de Q pertencentes ao conjunto de estados marcados (Q_m). Por fim, obtém-se a componente acessível e co-acessível (operação trim) do resultado da intersecção da linguagem da Planta (P) com I (Projeção inversa de E), com a finalidade de obter a linguagem alvo K contida em L_m.

Algoritmo 4.2 Algoritmo para obter a linguagem $K \subseteq (V^+ \times U)^*$

$Q \leftarrow \{q_0\}$; $Q_m \leftarrow \{q_0\}$; {Inicialização dos conjuntos que armazenam estados}
$Testa \leftarrow \emptyset$; $G_{pares} \leftarrow \emptyset$; {Inicialização dos conjuntos que armazenam pares do tipo (q,x) }
para $\forall u \in U$ **faça**
  $\gamma \leftarrow \eta/u$
  $\beta(q_0,\gamma) \leftarrow q'$
fim para
$G_{pares} \leftarrow (q',x_0)$
$Testa \leftarrow (q',x_0)$
enquanto $G_{pares} \neq \emptyset$ **faça**
  Selecione e remova um par $(q,x) \in G_{pares}$
  se $x \in X_m$ **então**
    $Q_m \leftarrow q$
  fim se
  para $\forall v \in \Sigma_E$ tal que $f(x,v)!$ **faça**
    $aux \leftarrow f(x,v)$
    se $\exists (q_a,x_a) \in Testa$ tal que $(q_a \in Q \wedge x_a = aux)$ **então**
      para $\forall u \in U$ **faça**
        $\gamma \leftarrow v/u$
        $\beta(q,\gamma) \leftarrow q_a$
      fim para
    senão
      Cria um novo estado q' em Q
      para $\forall u \in U$ **faça**
        $\gamma \leftarrow v/u$
        $\beta(q,\gamma) \leftarrow q'$
      fim para
      $G_{pares} \leftarrow (q',aux)$
      $Testa \leftarrow (q',aux)$
    fim se
  fim para
fim enquanto
$K = \text{trim}\ (P \cap I)$.

4.2 Algoritmos para o Controle Modular

Conforme visto anteriormente, para que se possa utilizar a abordagem modular para sistemas C/E não basta que as linguagens sejam não conflitantes; é necessário garantir a interconsistência entre as mesmas. Sendo assim, o algoritmo apresentado a seguir serve para testar a interconsistência entre as linguagens como formalmente apresentado pela Definição 3.2. Na seqüência apresenta-se o algoritmo para obter de forma sistemática a máxima sublinguagem interconsistente de K_1 que é interconsistente em relação a K_2.

4.2.1 Algoritmo para o teste da Interconsistência entre duas linguagens

Algumas definições são necessárias para o entendimento do Algoritmo 4.3.

- Seja $Sup\mathbb{C}_{VU}(K_i)$ a máxima linguagem *vu*-controlável contida em $K_i \subseteq (V \times U)^*$ $(i = 1,2)$.
  - O modelo que representa $Sup\mathbb{C}_{VU}(K_1)$ é representado por $Sup_{K1} = (X, \Sigma_{K1}, \delta, x_0, X_m)$, onde X é o conjunto de estados, Σ_{K1} é o alfabeto em $(V^+ \times U)^*$, δ é a função de transição, x_0 é o estado inicial e X_m é o conjunto de estados marcados.
  - O modelo que representa $Sup\mathbb{C}_{VU}(K_2)$ é representado por $Sup_{K2} = (Y, \Sigma_{K2}, \beta, y_0, Y_m)$, onde Y é o conjunto de estados, Σ_{K2} é o alfabeto em $(V^+ \times U)^*$, β é a função de transição, y_0 é o estado inicial e Y_m é o conjunto de estados marcados.
- O modelo que representa a intersecção das linguagens reconhecidas pelos autômatos Sup_{K1} e Sup_{K2} é obtido através da composição síncrona de dois autômatos com alfabetos comuns conforme segue. Define-se a composição síncrona de Sup_{K1} e Sup_{K2}, denotada $Sup_{K1} \parallel Sup_{K2}$, como o autômato $C = (Z, \Sigma, \gamma, z_0, Z_m)$ onde $Z = X \times Y$, $\Sigma = \Sigma_{K1} \cap \Sigma_{K2}$, $\gamma : Z \to Z$, $z_0 = (x_0, y_0)$ e $Z_m = X_m \times Y_m$. Seja $\sigma \in \Sigma$ e $z = (x,y) \in X \times Y$, então define-se

$$\gamma(z,\sigma) = \begin{cases} [\delta(x,\sigma), \beta(y,\sigma)] & \text{se } \delta(x,\sigma)! \text{ e } \beta(y,\sigma)! \\ \text{indefinida} & \text{caso contrário} \end{cases}$$

  A operação de composição síncrona de dois autômatos com alfabetos iguais é equivalente à intersecção das linguagens destes autômatos. O algoritmo para obter a composição síncrona de dois autômatos com alfabetos comuns é apresentado em [8].
- $V_{Sup_{K1}}(x)$ e $V_{Sup_{K2}}(y)$ representa o conjunto de eventos ativos $(v \in V^+)$ no estado $x \in X$ e $y \in Y$, respectivamente; $V_C(z)$ representa o conjunto de eventos ativos $(v \in V^+)$ no estado $z = (x,y) \in Z$.

Resumo do algoritmo:

Para cada estado $z = (x,y) \in Z$, o algoritmo verifica se o conjunto de eventos ativos em $z \in Z$ é diferente da intersecção dos conjuntos de eventos ativos no estado x em Sup_{K1} e y em Sup_{K2}. Caso afirmativo as linguagens $Sup\mathbb{C}_{VU}(K_1)$ e $Sup\mathbb{C}_{VU}(K_2)$ não são interconsistentes, e em caso contrário estas linguagens são interconsistentes.

Algoritmo 4.3 Algoritmo para o teste da Interconsistência entre duas linguagens

$Flag \leftarrow False$
$C = trim(Sup_{K1} \parallel Sup_{K2})$
para $\forall z = (x,y) \in Z$ **faça**
  se $(V_{Sup_{K1}}(x) \cap V_{Sup_{K2}}(y)) \neq V_C(z)$ **então**
    $Flag \leftarrow True$
    interrompa
  fim se
fim para
se $Flag = True$ **então**
  " As linguagens não são interconsistentes "
senão
  " As linguagens são interconsistentes "
fim se

4.2.2 Algoritmo para obter a máxima linguagem interconsistente

Além das definições apresentados no Algoritmo 4.3, as seguintes definições são necessárias para o entendimento do Algoritmo 4.4.

- *Mausestados* é um conjunto que armazena estados $x \in X$.
- *Aux* é um conjunto que armazena estados $z = (x,y) \in Z$.

Resumo do algoritmo: Dadas duas linguagens $Sup\mathbb{C}_{VU}(K_1)$ e $Sup\mathbb{C}_{VU}(K_2)$, o algoritmo para obtenção da máxima sublinguagem de $Sup\mathbb{C}_{VU}(K_1)$ que é interconsisntente em relação a $Sup\mathbb{C}_{VU}(K_2)$ detecta os estados de Sup_{K1} que levam a não interconsistência entre as linguagens e remove estes "maus estados". O procedimento é repetido até que não existam mais maus estados, ou seja, até que as linguagens sejam interconsistentes.

4.3 Conclusão

Neste capítulo apresentou-se uma série de algoritmos envolvidos na solução do problema de síntese de supervisores modulares para sistemas modelados através do paradigma de sistemas C/E. Vale

Algoritmo 4.4 Algoritmo para obter a máxima linguagem interconsistente

Função: *TestaInter(C)*
  $Mausestados \leftarrow \emptyset$
  $Aux \leftarrow \emptyset$
  para $\forall z = (x,y) \in Z$ **faça**
    se $(V_{Sup'_{K1}}(x) \cap V_{Sup_{K2}}(y)) \neq V_C(z)$ **então**
      $Mausestados \leftarrow Mausestados \cup \{x\}$
      $Aux \leftarrow Aux \cup \{z\}$
    fim se
  fim para
fim Função: *TestaInter(C)*

Função: *Principal*
  $Sup'_{K1} \leftarrow Sup_{K1}$
  $C = trim(Sup'_{K1} \parallel Sup_{K2})$
  TestaInter(C)
  enquanto $Mausestados \neq \emptyset$ **faça**
    Remove de Sup'_{K1} os estados $x \in Mausestados$
    Remove de C os estados $z \in Aux$
    C = trim (C)
    TestaInter (C)
  fim enquanto
  Sup'_{K1} = trim (Sup'_{K1}) {Sup'_{K1} é o modelo que representa a máxima sublinguagem de $Sup\mathbb{C}_{VU}(K_1)$ que é interconsistente em relação a linguagem $Sup\mathbb{C}_{VU}(K_2)$}
fim Função: *Principal*

salientar que estes algoritmos foram implementados na ferramenta **Grail**[1][15] com intuito de supri-la com um pacote de funções para o controle supervisório modular de sistemas C/E.

Por derradeiro, deve-se salientar que, em geral, os modelos obtidos para representar o comportamento lógico de sistemas de grande porte, bem como o de sistemas híbridos, são modelos complexos (com grande número de estados e transições). Além disso, sabe-se que, o aumento da complexidade do modelo leva a uma maior dificuldade na obtenção de supervisores, tornando imprescindível o desenvolvimento e implementação destes algoritmos de forma a automatizar o processo de obtenção destes supervisores.

[1]O grail é uma biblioteca de funções desenvolvida em C++, que permite a computação simbólica sobre máquinas de estado finitas, expressões regulares e linguagens

Capítulo 5

Exemplo

5.1 Introdução

Neste capítulo apresenta-se um exemplo que envolve um problema prático de controle supervisório de forma a ilustrar a metodologia apresentada anteriormente. Vale ressaltar que este mesmo exemplo foi utilizado no Capítulo 2(Seção 2.3) para ilustrar a metodologia de modelagem e será retomado para apresentar a solução do problema pelo uso da ferramenta computacional Grail através das funções que implementam os algoritmos desenvolvidos ao longo deste trabalho, ilustrando assim a utilização das mesmas.

Este exemplo será resolvido tanto na abordagem monolítica quanto na modular, de forma a comparar os resultados obtidos nestas duas abordagens. Vale lembrar ainda que um exemplo de sistema híbrido que inclua um problema de controle supervisório modular foi apresentado em [13, 14].

5.2 Descrição do Problema

A célula de manufatura estudada é composta por uma mesa circular de três posições, onde são efetuadas operações de furo sobre peças metálicas, e de mais três dispositivos operacionais: a esteira de entrada, a furadeira e o manipulador robótico conforme ilustrado na Figura 5.1. O funcionamento da mesa giratória é comandado por um controlador lógico programável (CLP) sendo que o programa de controle fornecido pelo fabricante executa a seguinte seqüência de passos:

1. A esteira gira até que uma peça seja posicionada em P1;
2. A mesa gira 120^o;

3. A peça é furada;

4. A mesa gira 120^o;

5. O manipulador robótico retira a peça da mesa.

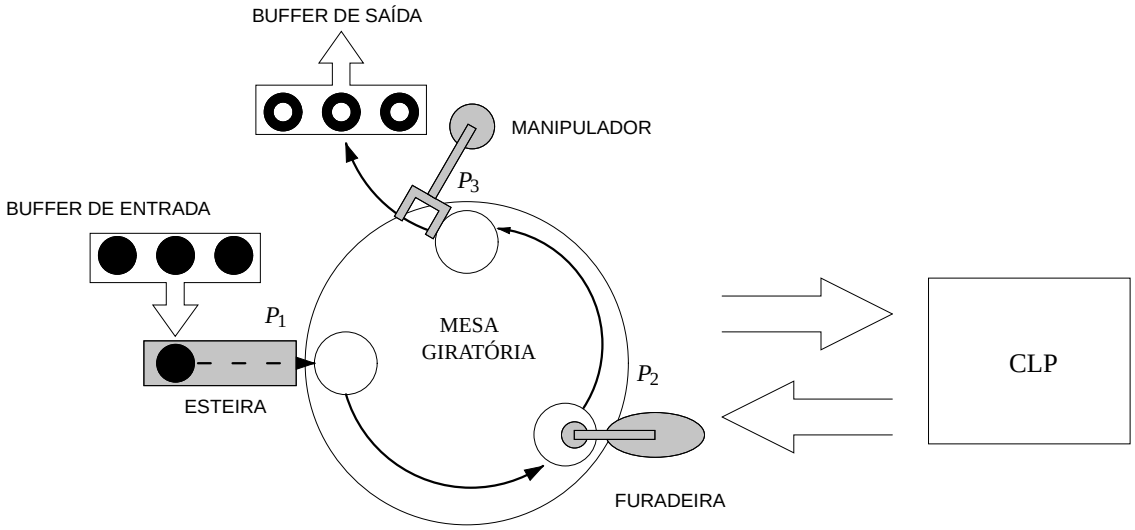


Figura 5.1: Sistema com mesa giratória

Observe que o programa de controle original da mesa permite operar em seqüência apenas uma peça por vez, ou seja, a esteira só pode ser acionada novamente depois que o manipulador retirar a peça da mesa. Esta restrição na lógica de controle evita os problemas que podem ocorrer na operação de múltiplas peças em paralelo, entre os quais se destacam:

- operar a esteira, a furadeira, ou o manipulador enquanto a mesa estiver girando;
- sobrepor peças em P1;
- girar a mesa sem que as peças em P2 e P3 tenham sido furadas ou retiradas, respectivamente;
- furar ou acionar o manipulador sem peças nas posições P2 e P3, respectivamente;
- furar duas vezes a mesma peça;
- girar a mesa sem nenhuma peça.

Entretanto, esse modo de funcionamento é muito pouco eficiente, visto que a esteira, a furadeira, e o manipulador passam a maior parte do tempo parados, enquanto poderiam estar operando em paralelo. O problema a ser resolvido é desenvolver uma lógica de controle que garanta uma maior eficiência da mesa giratória.

5.3 Modelagem

De acordo com a metodologia proposta em [6], o primeiro passo para a modelagem de um sistema consiste em encontrar os subsistemas envolvidos no sistema como um todo. Na modelagem desse sistema são quatro os subsistemas envolvidos: esteira, mesa giratória, furadeira e o manipulador robótico. O segundo passo consiste em verificar o relacionamento dos subsistemas e aplicar a operação de conexão sobre os subsistemas relacionados. Vale ressaltar que ambos os passos foram apresentados na Seção 2.3. Desta forma pode-se apresentar o modelo da planta livre.

5.3.1 Modelagem da Planta

Como apresentado na Seção 2.3, após obter os modelos dos subsistemas, pode-se obter o modelo C/E da planta livre $\mathcal{P}$. Com o propósito de simplificar o modelo da planta, facilitando assim o entendimento do exemplo, o autômato da planta livre modela o comportamento do sistema atuando nos moldes de uma restrição que impede a mesa de girar enquanto a esteira, a furadeira ou o robô estiverem operando. Esta restrição consiste em uma restrição do tipo "estado proibido".

O autômato C/E que representa a planta possui 9 estados e 112 transições e é ilustrado na Figura 5.2.

Para facilitar o entendimento do modelo que representa o comportamento do sistema algumas observações serão feitas.

- a notação $\bullet/u_i$ indica transições com todos os eventos associados à condição u_i. Assim, a notação $\bullet/u_5$ indica as transições $v_1/u_5, v_2/u_5, v_3/u_5, v_4/u_5$;
- a notação $v_i/\bullet$ indica transições com o evento v_i, associadas à qualquer condição. Assim, a notação $v_2/\bullet$ indica as transições $v_2/u_5, v_2/u_6, v_2/u_7, v_2/u_8, v_2/u_{12}, v_2/u_{13}, v_2/u_{14}, v_2/u_{15}$.

Para que se possa utilizar o pacote de funções para o controle supervisório de sistemas C/E desenvolvido para o Grail, é necessário representar o autômato C/E que modela a planta no formato aceito pelo Grail. A representação de máquina de estados finitas (FM) no Grail consiste em um lista de instruções armazenada em um arquivo **ASCII**, sendo que cada instrução é uma tripla formada por um estado fonte, um símbolo e um estado destino. Em especial, na representação de um autômato C/E no Grail os símbolos são pares ordenados de inteiros, sendo que o primeiro elemento indica o evento *v* e o segundo a condição *u*. Os estados iniciais devem ser indicados através da pseudo-instrução (START) |- e os estados finais devem ser indicados por -| (FINAL). Além disso, o número inteiro zero é reservado para indicar o evento de inicialização η.

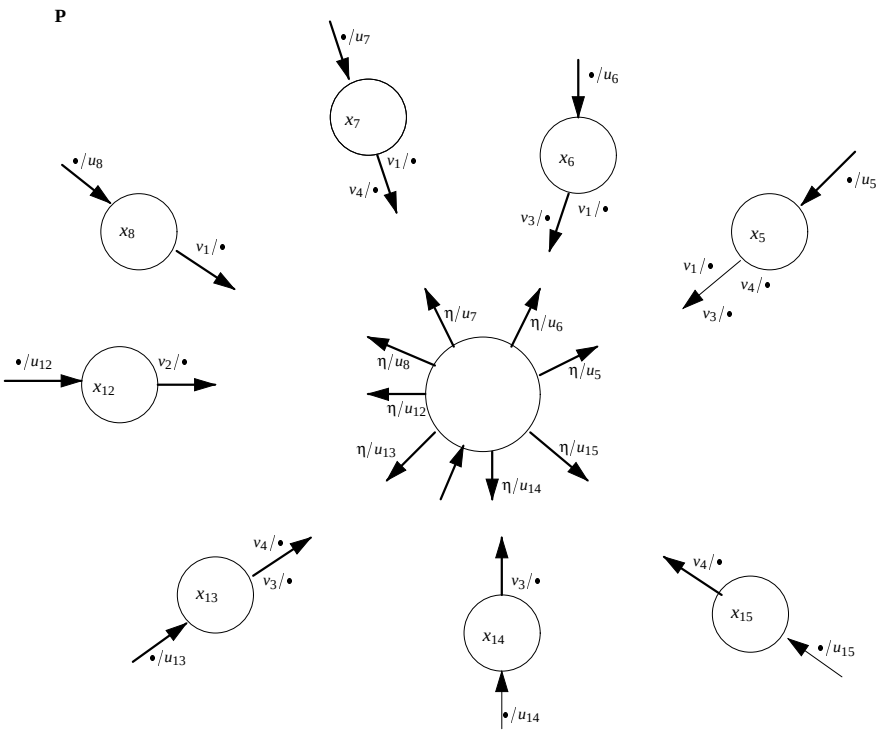


Figura 5.2: Autômato que representa o comportamento livre da mesa giratória

Seja por exemplo um autômato C/E que reconhece a linguagem marcada $\eta u_1 \circ v_1 u_2$. A Figura 5.3 mostra este autômato e a sua forma de representação no Grail, sendo que os pares [0, 1] e [1, 2] são usados para representar ηu_1 e $v_1 u_2$, respectivamente.

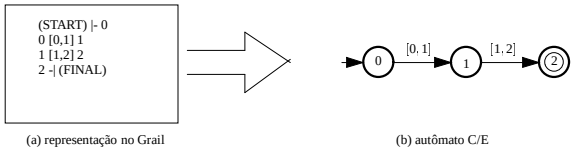


Figura 5.3: Exemplo de um autômato C/E representado no **Grail**

A ferramenta **Grail** pode ser obtida na página http//www.das.ufsc.br/∼cury/ensino-seds.html, onde também se encontram instruções de instalação da ferramenta. Para a visualização gráfica dos resultados utiliza-se a ferramenta **Graphviz**, a qual pode ser encontrada na página http://www.researh.att.com/sw/tools/graphviz.

Os pacotes para o controle supervisório (abordagens monolítica e modular) de sistemas C/E desenvolvidos para o **Grail** encontram-se na página http://www.das.ufsc.br/∼dlr. Seguindo a filosofia de interfaceamento do Grail, as funções implementadas estão disponíveis como comandos no *prompt* do sistema, uma vez a ferramenta instalada. Não há um executável específico para invocar a ferramenta para controle supervisório, e usa-se qualquer janela de terminal do sistema para acessar suas funções, seja um *shell*, *prompt*, *telnet* ou *ssh*. Os arquivos que armazenam os dados dos modelos são

arquivo de texto ASCII contendo informações em dados reconhecíveis pela ferramenta, e que podem ser criados e modificados usando qualquer editor de texto.

5.3.2 Modelagem das especificações

Com o objetivo de sintetizar um programa de controle para o CLP que garanta o funcionamento correto da mesa, uma série de restrições foram impostas para o sistema. Estas restrições, dadas em termos de seqüências de eventos, são as seguintes:

- Impedir que a mesa gire à toa (Figura 5.4), isto é sem ao menos uma peça em P1 ou uma peça furada em P2 ;

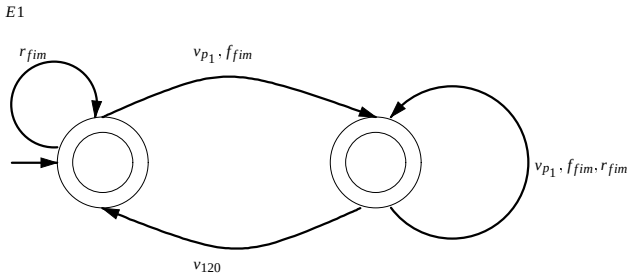


Figura 5.4: Restrição que impede a mesa de gira à toa

- Restrição que controla o fluxo de peças entre P1 e P2. Esta restrição é mostrada na Figura 5.5 e serve para evitar as seguintes ações:
  - sobrepor peças em P1;
  - furar sem peça em P2; e
  - girar com peça bruta em P2.

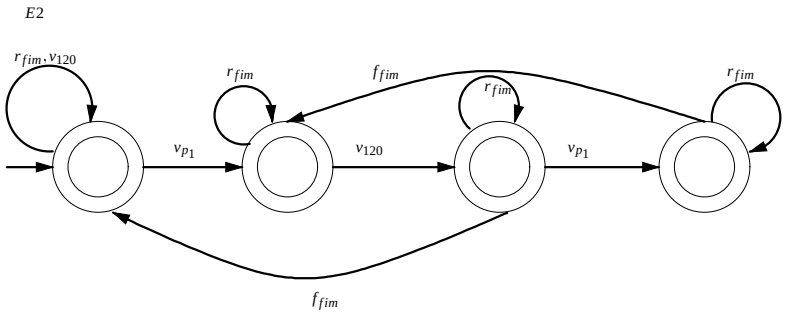


Figura 5.5: Restrição que controla o fluxo de peças entre P1 e P2

- Restrição que controla o fluxo de peças entre P2 e P3. Esta restrição é mostrada na Figura 5.6 e evita as seguintes ações indesejadas:

– furar duas vezes a mesma peça;

– acionar o robô sem peça em P3; e

– girar com peça em P3.

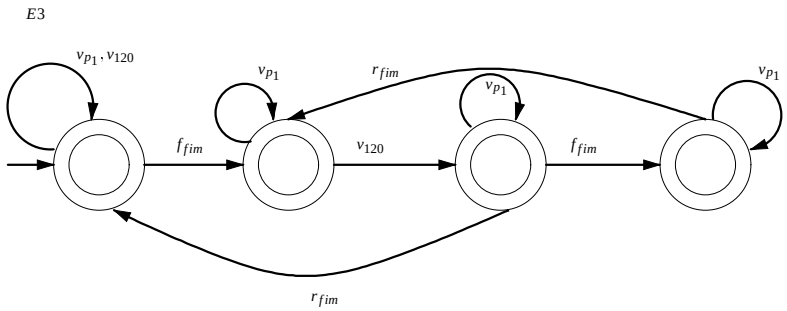


Figura 5.6: Restrição que controla o fluxo de peças entre P2 e P3

Novamente, deve-se representar estas especificações no formato aceito pelo Grail. Para exemplificar, a Figura 5.7 apresenta a especificação da Figura 5.5 no formato do Grail e visualizada no Graphviz. Note que os símbolos para representar os eventos são números inteiros, por exemplo o número inteiro "1" representa o evento v_{P_1} (ver Tabela 5.1) .

Tabela 5.1: Simbologia dos eventos no **Grail**

número inteiro	evento	Descrição
1	v_{P_1}	Final de operação da esteira
2	v_{120}	Final de giro da mesa
3	f_{fim}	Final de operação da furadeira
4	r_{fim}	Final de operação do manipulador

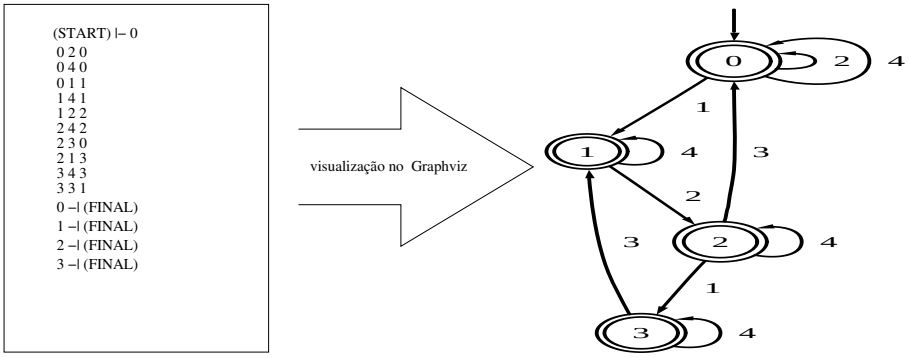


Figura 5.7: Especificação da Figura 5.5 no **Grail** e vizualizada no **Graphviz**

5.4 Aplicação do Controle Supervisório

Após encontrar o modelo da planta livre e das restrições para o sistema, deve-se projetar um supervisor C/E para supervisioná-la. O relacionamento da planta livre com o supervisor pode ser examinado na Figura 5.8.

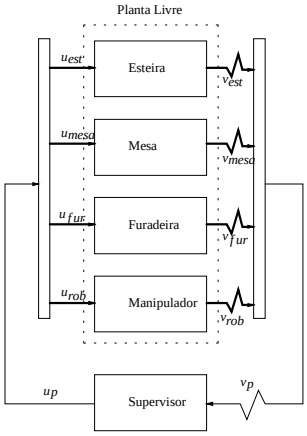


Figura 5.8: Relacionamento entre a planta livre e o Supervisor

5.4.1 Abordagem Monolítica

Após modelar as restrições é possível encontrar uma especificação $E \subseteq P_V(L_m)$ que modela o comportamento desejado para o sistema em malha fechada. Para isto, realiza-se a intersecção entre todas as restrições, ou seja, $E = E1 \cap E2 \cap E3$. A Figura 5.9 mostra o autômato que representa a especificação global $E \subseteq V^*$. Deve -se salientar que, no intuito de simplificar a notação empregada, utilizar-se-á o mesmo nome para a linguagem e o autômato que a reconhece. Assim, a especificação E é representada por uma FM denotada pelo seu próprio nome.

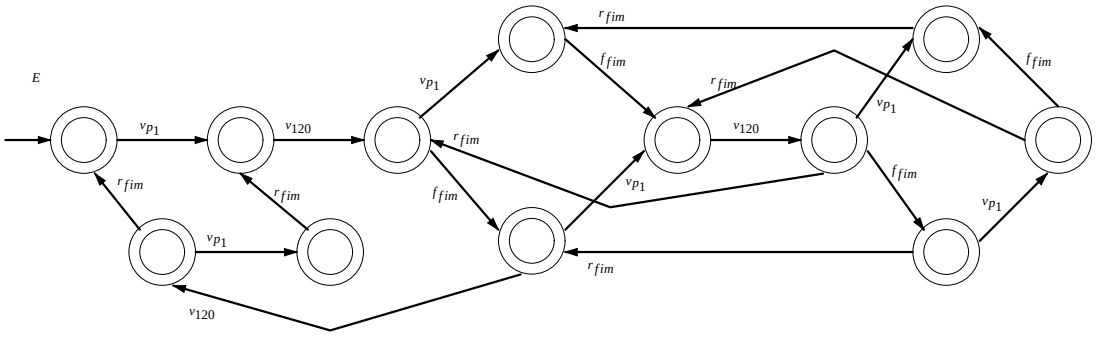


Figura 5.9: Especificação global

Para realizar a intersecção das especificações E_i $(i = 1,2,3)$ utiliza-se uma função do Grail cha-

mada **fmcross**, conforme mostrado a seguir. Todas as funções utilizadas para a solução deste problema de controle supervisório de sistemas C/E são executadas no *prompt* do Windows.

c:\> fmcross E1 E2 > E12

c:\> fmcross E12 E3 > E

O símbolo >, usado acima, representa o redirecionamento da saída de um programa, originalmente direcionado para a saída padrão (visor). Por exemplo, no segundo caso acima, redireciona-se a saída do comando **fmcross E12 E3** para o arquivo **E**. Esta FM modela a especificação global ($E = E1 \cap E2 \cap E3$).

Conhecendo-se a especificação E, obtém-se a especificação equivalente $K = P_V^{-1}(E) \cap L_m$ através da função **iok**, a qual implementa o Algoritmo 4.2. A utilização desta função é como segue:

c:\> iok E P > K

Onde:

- E contém a FM que modela a especificação global;
- P contém a FM que representa o comportamento seqüencial lógico do sistema C/E;
- K é o autômato (FM) que representa a especificação equivalente.

Uma nota sobre o prefixo das funções: seguindo a notação do **Grail**, o prefixo da função representa o objeto principal manipulado pela mesma [15]. Assim, são definidas funções com prefixo **fm** para FMs, prefixo **io**[1] são definidas para sistemas tipo entrada/saída. Uma particularidade para as funções de controle supervisório de sistemas C/E é que quando invocadas sem argumento ou de forma errada, apresenta-se na saída padrão um texto de ajuda de utilização.

O próximo passo consiste em obter o supervisor SED que garanta o cumprimento da especificação E. Este supervisor é obtido com ajuda do algoritmo que calcula a máxima linguagem *vu*-controlável e prefixo fechada, o qual foi desenvolvido e implementado em trabalho anterior [9]. Segundo a Proposição 2.1, existe um supervisor C/E equivalente ao supervisor SED encontrado. Para obtê-lo basta aplicar o algoritmo para encontrar a realização de estado discreto do supervisor C/E, desenvolvido em [6].

Para obter o supervisor SED utiliza-se a função **iosupc**, conforme ilustrada a seguir:

c:\> iosupc P K > S

Onde

[1] do inglês original, *input/output*, a denominar um sistema condição/evento

- S é a saída da função, uma máquina de estado finita (FM) que representa a máxima linguagem *vu*-controlável contida em K.

O autômato que representa o supervisor é mostrado na Figura 5.10. O comportamento da planta sob ação do supervisor é como segue. Na inicialização da planta ocorre o evento $v_0 = [\eta, \eta, \eta, \eta]$ e o supervisor aplica a condição u_8 que indica o comando de ligar a esteira, transitando para o estado x_8. No estado x_8 o supervisor percebe o evento v_1 (chegada de peça em P_1) e aplica a condição u_{12}, ou seja, liga a mesa, transitando para o estado x_{12}. No estado x_{12} o supervisor lê o evento v_2 (fim de giro da mesa) e aplica a condição u_6, resultando no comando para ligar a esteira e a furadeira, transitando para o estado x_6 e assim, sucessivamente.

É importante ressaltar que no intuito de otimizar a operação da mesa giratória, foi adicionado um mecanismo de escolha ao supervisor. Assim, se após a ocorrência de um determinado evento na planta o supervisor puder escolher mais de uma condição a ser aplicada à planta, este mecanismo escolhe a condição que habilita o funcionamento simultâneo do maior número de equipamentos. Entretanto, caso existam condições que habilitem o funcionamento do mesmo número de equipamentos, então esta escolha é feita de forma aleatória. Note ainda que, embora no resultado fornecido pelo ferramenta para controle supervisório de sistemas C/E os estados e as etiquetas sejam representadas por números, para facilitar o entendimento do exemplo os estados, condições e eventos do autômato do supervisor são representados por elementos dos vetores X_P, U_P e V_P, respectivamente (ver Seção 2.3).

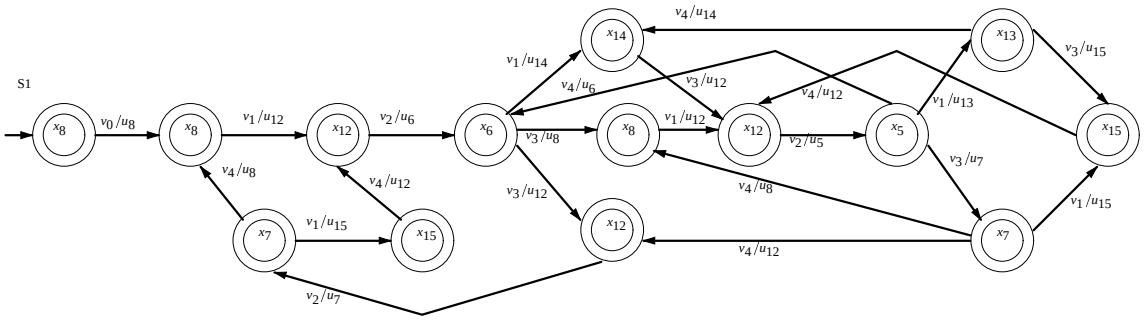


Figura 5.10: Autômato do Supervisor SED

Por fim, pode-se obter a máxima linguagem *v*-controlável contida na especificação $E \subseteq V^*$ através da função **ioproj** (implementação do Algoritmo 4.1). A utilização desta função é como segue:

c:\> ioproj S

Resumidamente, os passos para resolver o problema de controle supervisório de sistemas C/E no **Grail** e na ferramenta que implementa os algoritmos desenvolvidos neste trabalho são os seguintes:

1. Encontrar o autômato da planta;

2. Obter a especificação global: definir as FMs com as especificações e, se existirem duas ou mais especificações, utilizar a função **fmcross** para encontrar a especificação global;

3. Encontrar a especificação equivalente, através da função **iok**;

4. Utilizar a função **iosupc** para encontrar o supervisor SED;

    - Pode-se verificar ainda a máxima linguagem *v*-controlável, através da função **ioproj**.

5.4.2 Abordagem Modular

O mesmo exemplo usado anteriormente vai ser tratado agora através da abordagem modular. Nesta abordagem, projeta-se supervisores para cada especificação em separado, e aplica-se estes supervisores combinados, de modo a se atender a especificação global no sistema resultante em malha fechada. O projeto de cada um dos supervisores modulares é desenvolvido conforme apresentado na Seção 5.4.1. Vale salientar que, em virtude do seu tamanho excessivo (número de estados e transições) os autômatos dos supervisores modulares não serão mostrados aqui.

Após obter o supervisores modulares, testa-se a interconsistência entre as linguagens. Neste caso tal condição é verificada e portanto os supervisores podem ser implementados de forma modular. Vale observar que o teste da interconsistência deve ser aplicado entre todos os supervisor (dois a dois).

A função que testa a interconsistência entre as linguagens foi denominada **iointer** (implementação do Algoritmo 4.3), e é utilizada da seguinte maneira:

c:\> iointer S1 S2

Onde:

- S1 contém o autômato que representa a linguagem do supervisor projetado para cumprir com a especificação da Figura 5.4;

- S2 contém o autômato que representa a linguagem do supervisor projetado para cumprir com a especificação da Figura 5.5.

Pode-se, então, obter a linguagem resultante da ação conjunta dos supervisores, através da função **fmcross**. A Figura 5.11 ilustra o autômato que representa a ação conjunta dos supervisores. Comparando este resultado com o resultado obtido anteriormente, constata-se que a solução obtida através da abordagem modular é igual àquela obtida utilizando-se a abordagem monolítica.

Vale ressaltar que caso as linguagens não fossem interconsistentes (linguagens representadas por S1 e S2) poderíamos fixar uma das linguagens (por exemplo, a linguagem modelada por S2) e usar a

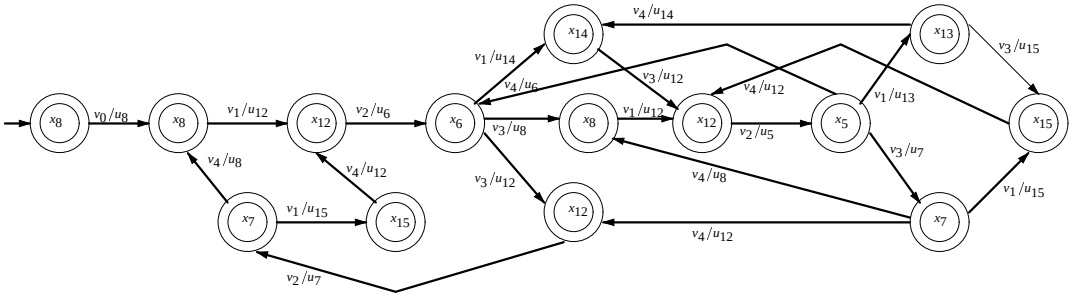


Figura 5.11: Autômato que representa a ação conjunta dos supervisores

função **iosupi** (implementação do algoritmo 4.4) para obter a máxima linguagem interconsistente em relação a linguagem fixada. Utiliza-se então, de forma alternada, as funções **iosupi** e **iosupc** até que a linguagem modelada por S1 seja *vu*-controlável e interconsistente em relação a S2.

A função que calcula a máxima linguagem interconsistente é utilizada da seguinte maneira:

c:\> iosupi S1 S2 > S1'

Onde:

- S1' conteria o autômato que representa a máxima linguagem interconsistente em relação a linguagem modelada por S2.

5.5 Conclusão

Neste capítulo, utilizou-se um exemplo prático para ilustrar a resolução de um problema de controle supervisório de sistemas C/E. A resolução do exemplo se deu tanto na abordagem monolítica quanto na modular e foi realizada com o auxílio do pacote desenvolvido para controle supervisório de sistemas C/E para a ferramenta Computacional **Grail**.

Capítulo 6

Conclusão e Perspectivas

O presente trabalho teve como objetivo geral o desenvolvimento e implementação de algoritmos envolvidos na síntese de supervisores modulares para sistemas modelados através do paradigma de sistemas condição/evento (C/E).

Uma das grandes motivações para o uso da abordagem de sistemas C/E, é que ela permite uma forma de se definir sistemas em tempo continuo através da interconexão de subsistemas com sinais de entrada e saída discretos. Assim, ela adapta-se perfeitamente tanto à modelagem de sistemas a eventos discretos, quanto à modelagem de sistemas híbridos. Além disso, ela possibilita que a modelagem destas classes de sistemas seja baseada em diagramas de blocos e fluxos de sinais, como é feito normalmente na teoria de sistemas. Para esta primeira classe de sistemas, acredita-se que este tipo de paradigma de modelagem torna-se muito mais natural e intuitivo que aqueles oferecidos por formalismos estritamente discretos, tais como autômatos e linguagens formais.

A abordagem de controle modular, de uma forma geral, possui duas grandes motivações quando comparada com a abordagem monolítica: diminuir a complexidade computacional e aumentar a flexibilidade em caso de mudanças: por exemplo, se uma especificação relativa a uma subtarefa é alterada pode-se reprojetar apenas o supervisor referente àquela especificação.

Tendo em vista que os modelos obtidos para representar o comportamento lógico de sistemas a eventos discretos de grande porte, bem como o de sistemas híbridos, são modelos complexos (com grande número de estados e transições), a viabilidade do uso da metodologia de síntese proposta passa forçosamente pela disponibilidade de ferramentas computacionais que a implementem. Além disso, sabe-se que, o aumento da complexidade do modelo leva a uma maior dificuldade na obtenção de supervisores, tornando indispensável a automatização do processo de obtenção destes supervisores.

As principais contribuições deste trabalho são:

- O desenvolvimento de algoritmos envolvidos na solução do problema de síntese de supervisores

para sistemas C/E sob o ponto de vista da teoria de controle supervisório de SEDs (abordagens monolítica e modular) ;

- A implementação dos algoritmos desenvolvidos em um ferramenta computacional para controle supervisório de sistemas C/E;
- A aplicação do pacote desenvolvido para a solução de um problema prático de síntese de supervisores.

Por fim, deve salientar que a ferramenta desenvolvida foi utilizada em outros trabalhos de pesquisa desenvolvidos no âmbito do DAS, a saber, no trabalho de doutorado do aluno André Bittencourt Leal [11] e no trabalho de mestrado do aluno Danilo de Paula e Silva [16].

Quanto a perspectivas futuras, pretende-se aperfeiçoar os programas desenvolvidos, com o intuito de agrupar as funções à ferramenta Grail e analisar a complexidade computacional dos algoritmos desenvolvidos.

Referências Bibliográficas

[1] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Massachussets, 2nd edition, 1999.

[2] J. E. R. Cury and B. H. Krogh. Synthesizing supervisory controllers for hybrid systems. *Journal of the Society of Instrument and Control Engineers (SICE)*, 38(3):161–168, March 1999.

[3] J. E. R. Cury, B. H. Krogh, and T. Niinomi. A methodology for the design of supervisory controllers for a class of hybrid systems. *Anais do XI Congresso Brasileiro de Automática, CBA 1996*, 1996.

[4] J. E. R. Cury, B. H. Krogh, and T. Niinomi. Synthesis of supervisory controllers for hybrid systems based on approximating automata. *IEEE Transactions on Automatic Control, Special Issue on Hybrid Control Systems*, 43(4):564–568, April 1998.

[5] M. H. de Queiroz. *Controle Modular Local para Sistemas de Grande Porte*. Dissertação (mestrado), Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC, março 2000.

[6] T. R. Garcia. *Controle Supervisório de Sistemas a Eventos Discretos: Uma Abordagem por Modelo Condição/Evento*. Dissertação (mestrado), Progama de Pós Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC, março 2002.

[7] T. R. Garcia and J. E. R. Cury. Controle supervisório de sistemas condição/evento. *Anais do XIV Congresso Brasileiro de Automática - CBA 2002*, pages 449–454, setembro 2002.

[8] J. M. E. González. *Aspectos de Síntese de Supervisores para Sistemas a Eventos Discretos e Sistemas Híbridos*. Tese (doutorado), Programa de Pós Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, Brasil, abril 2000.

[9] J. M. E. González, A. E. C. da Cunha, J. E. R. Cury, and B. H. Krogh. Supervision of event-driven hybrid systems: Modeling and synthesis. In *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science, pages 247–260, Rome, Italy, January 2001. Springer-Verlag.

[10] B. H. Krogh and S. Kowalewski. State feedback control of condition/event systems. *Mathematical and Computer Modelling*, 23(11/12):161–173, 1996.

[11] A. B. Leal. *Controle Supervisório Modular de Sistemas Híbridos*. Tese (doutorado), Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC, 2004.

[12] A. B. Leal and J. E. R. Cury. Controle modular de sistemas condição/evento. *Anais do XIV Congresso Brasileiro de Automática, CBA 2002*, pages 455–460, setembro 2002.

[13] A. B. Leal and J. E. R. Cury. Modular supervision of hybrid systems: A DES approach. A ser apresentado no 7th Workshop on Discrete Event Systems - WODES'04, 2004.

[14] A. B. Leal and J. E. R. Cury. On the existence of optimal solutions for the modular supervisory control of hybrid systems. Submetido ao 43rd IEEE Conference on Decision and Control - CDC, 2004.

[15] D. R. Raymond and D. Wood. GRAIL: A C++ Library for Automata and Expressions. *Journal of Symbolic Computation*, 17(4):341–350, April 1994.

[16] D. P. Silva. *Modelagem, Verificação e Controle Supervisório de Sistemas Híbridos Aplicados em uma Planta Piloto*. Dissertação (mestrado), Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC, maio 2004.

[17] R. S. Sreenivas and B. H. Krogh. On condition/event systems with discrete state realizations. *Discrete Event Dynamic Systems: Theory and Applications*, 1(3):209–236, 1991.

[18] C. R. C. Torrico. *Implementação de Controle Supervisório de Sistemas a Eventos Discretos Aplicado a Processos de Manufatura*. Dissertação (mestrado), Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC, 1999.

[19] W. M. Wonham. *Notes on Control of Discrete-Event Systems*. Systems Control Group, Department of Electrical & Computer Engineering, University of Toronto, Toronto, Canada, 1999.

[20] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, May 1987.

[21] R. M. Ziller. *A Abordagem Ramadge-Wonham no Controle de Sistemas a Eventos Discretos: Contribuições à Teoria*. Dissertação (mestrado), Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC, Outubro 1993.